A METHODOLOGY TO DEVELOP MICROPROCESSOR BASED I/O CONTROLLERS

A Thesis Submitted
in Partial Fulfilment of the Requirements
for the Degree of
MASTER OF TECHNOLOGY

By

PADMA KRISHNA RAO

to the
DEPARTMENT OF ELECTRICAL ENGINEERING
INDIAN INSTITUTE OF TECHNOLOGY, KANPUR
AUGUST, 1981

EE-1981-M-RAO-MET

CERTIFICATE

Certified that the work entitled, " A METHODOLOGY TO DEVELOP MICROPROCESSOR BASED I/O CONTROLLERS" by Mr. Padma Krishna Rao has been carried out under my supervision and has not been submitted elsewhere for the award of a degree.

(Dr. V. Rajaraman)
Professor of Computer Science
and
Electrical Engineering
Department of Electrical Engineering
Indian Institute of Technology, KANPUR.

KANPUR
August 10th, 1981.

ACKNOWLEDGEMENTS

I wish to express my deep sense of gratitude to Professor V. Rajaraman for the constant encouragement and guidance provided by him throughout the course of my thesis.

I would like to thank the faculty and my friends, for the co-operation extended to me. In particular, I would like to thank Mr. C. G. Ganesh for his help during the all stages of my thesis work.

Acknowledgements are due to Mr. R. B. Singh and Mr. S. S. Pethkar for their excellent typing work.

Kanpur
August , 1981.

- Padma Krishna Rao

CONTENTS

CHAPTER 1

INTRODUCTION

Data handling by a computer often involves the transfer of data between the memory and I/O devices. Some I/O devices can transmit a large amount of data in a short time. If the CPU had to process every character separately, a great deal of CPU time would be wasted. To avoid tying up the CPU for long periods of time on I/O, most medium and large computers have one or more specialized, low-cost I/O processors. Because the I/O is performed by these special processors, the CPU is available to spend most of its time on more difficult computations. The I/O processor along with the other hardware used to perform I/O operation is called I/O controller. With the advent of high performance microprocessors, it is possible to control I/O operation of low to medium speed I/O devices with the microprocessors. The present work involves developing and evaluating Microprocessing Unit-Based I/O Controllers.

1.1 MPU-BASED I/O CONTROLLERS : Designing the microprocessor into the controlling system allows hardware/software tradeoffs to be made to satisfy the specific system requirements. For example, in high speed devices, additional logic might be required if the desired data

transfer rate is to be met even though the microprocessing unit (MPU) is only used for device control. In low speed devices almost all of the control functions may be assumed by the MPU with little external hardware. It is assumed that the best tradeoff occurs by minimizing the hardware in the controller. High performance microprocessors provide an efficient means for controlling the high speed devices and in the lower speed applications, additional functions can be combined with the controller function to produce a more cost-effective system.

1.2 OBJECTIVE OF THE THESIS :

The objective of the thesis is to develop a methodology to develop MPU-based I/O controllers. As is generally the case with MPU based designs, there are numerous ways to solve the problem. The methodology that is selected is to satisfy the following two basic objectives :

1. To develop as generalized controller as possible. By "generalized" it is meant that the same controller could be used to handle different types of devices.
2. Use minimum external hardware.

An MPU-based I/O controller basically consists of the following modules :

1. the basic MPU system
2. the external hardware and
3. the software control modules.

After making a detailed study of the characteristics of different kinds of I/O devices, a generalized MPU system is developed which can be used to handle different kinds of I/O devices. In the case of the external hardware and the software control modules, it is found that they are very much dependent on the type of device and it is not feasible to make absolute generalization in these modules. However, these modules can be generalized if they are restricted to handle same kind of I/O devices but having different speeds . For example, all tape devices having different speeds can be controlled with common external hardware and software control modules. Some perameters, whose values are dependent on the device speed and characteristics, are passed to these modules to make them adaptive to handle the particular speed of the device, as dictated by the parameters.

The MPU-based I/O controllers are developed for the following three different kinds of devices to illustrate the various steps involved in designing MPU-based I/O controllers :

1. Low speed printer
2. Tape cassette system
3. Flpppy disk system.

It is assumed that the microprocessor is fast enough such that most of the control functions are carriedout by the software and minimum external hardware is used. The

generalized software control modules are developed for each device in PL/M, a higher language, whose compiler is supplied by the INTEL corporation. These software control modules are then evaluated to find out their capability of handling high speed devices.

1.3 OVER-VIEW OF THE THESIS :

Chapter 2 :- In this chapter MPU-based system design techniques are explained. The common characteristics of I/O devices are studied. A generalised MPU system is proposed for controlling the I/O devices. The various steps involved and the considerations that are to be taken into account while developing MPU-based I/O controllers are explained in detail.

Chapter 3 :- In this chapter MPU-based I/O controllers are developed for the following I/O devices following the methodology described in chapter 2 :

1. Low speed printer
2. Tape cassette system
3. Floppy disk system

Chapter 4 : In this chapter, the procedure for evaluating control routines is explained. The software control routines developed for the three devices in chapter 2 are evaluated and their capability of handling high speed devices is checked.

Chapter 5 : The thesis is concluded with a critical review of the present work and suggestions for future work.

CHAPTER 2

MPU-BASED I/O CONTROLLERS DESIGN TECHNIQUES

Development of a microprocessor-based system is similar in most respects to the design of conventional SSI/MSI systems. Both approaches must include the steps shown in Figure 2.1 : specification, system flow charts, hardware design and test and debug. However, the MPU based design adds another dimension. As indicated in figure 2.2 , the designer also has the option of software to consider and must decide whether each task is best done using the conventional approach or the software approach.

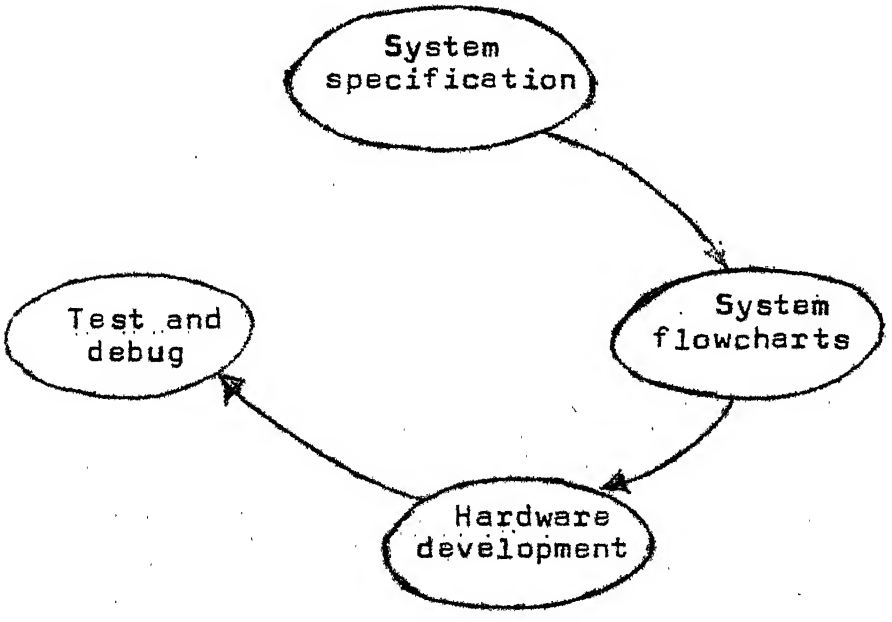


Fig. 2.1 : Conventional Design Cycle

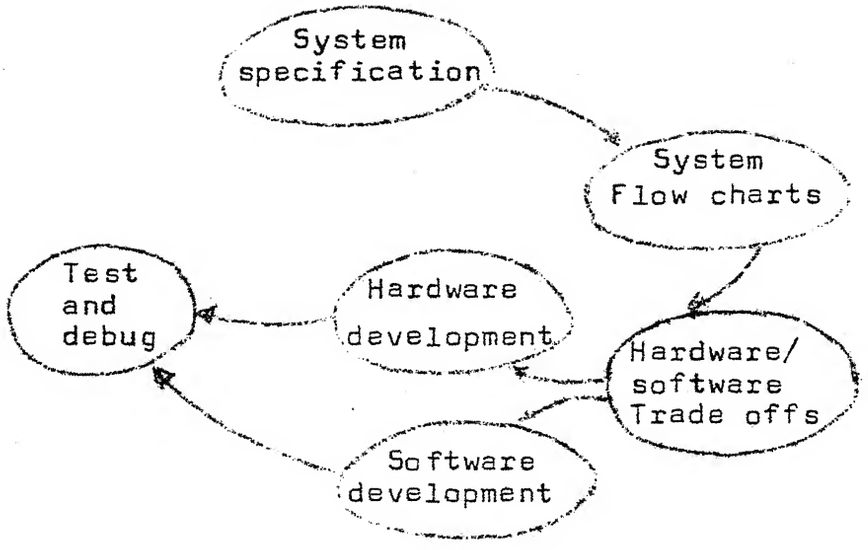


Fig. 2.2 : MPU- Based Design Cycle

The additional decisions should not be construed as an additional burden; they are in fact the key to developing the most cost-effective system. Study of the system specification will often provide indications as to the best approach. In addition to the MPU and its associated memory and interface devices, three additional elements are present in a typical system design: (1) the actual peripheral equipment that is dictated by the system specification; (2) any conventional electronics required to control the peripherals; (3) the "intelligence" that enables the MPU to perform the required control and data processing functions.

In an MPU based design, "intelligence" refers to the control program, a sequence of instructions that will guide MPU through the various operations it must perform. The

program, usually called "software" at this point, is then stored in read-only memory that can be accessed by the MPU during operation, thus becoming the system's intelligence. Once in memory the program is often called "firmware". However, it is common to find the terms software and firmware used interchangeably in this context.

When peripherals satisfying the system requirement have been selected, the designer can begin to consider tradeoffs that will result in the most cost-effective system .The presence of the MPU and control program in the system provides the designer with tradeoff opportunities not available in conventional designs.

In the next few sections, the characteristics of the I/O devices and the functions of the I/O controllers are studied in detail to get a clear idea about the specifications of I/O device controllers.

2.1 I/O DEVICE CHARACTERISTICS :

Literally hundreds of kinds of I/O devices are available to-day. A few of the more common ones are listed below :

- Line printers
- Magnetic tape storage devices
- Floppy Disk Systems
- Paper Tape Readers
- Paper Tape Punches
- Card Readers
- Card Punches
- Teletype Writers

S. No.	DEVICE NAME	TYPE OF DEVICE	STORAGE MEDIA MATERIAL	MOVABLE?	R/W head MOVABLE?	Basic functions of device controller
1	PRINTER	OUTPUT	PAPER	YES	Depend on the type of printer	The basic function of the controller is to print a line of text on the printer paper. The paper is divided into lines and each line is divided into several columns. A column in a line occupies a character. The controller shoulc scan the input text before printing a line. According to the input text, appropriate characters should be printed in each column of the line. After printing a line, the controller should advance the paper before starting the printing of the next line of the text.
2.	Magnetic tape device	INPUT/ output	Magnetic tape	Yes	No	The functions of the controller are 1. to bring the desired data block under R/W head by moving the tape and 2. to read/write a block of data. The first function includes rewinding the tape i.e. bringing the first data block under the R/W head.
3.	CARD READER	INPUT	A deck of cards	Yes	No	The function of the controller is to read characters from a card. The controller must pick up the next card from the input deck and read the characters from it by moving it past the R/W head.
4.	FLOPPY DISK	INPUT/ OUTPUT	Magnetic dish	Yes	Yes.	The functions of the controller are to position the R/W head on the desired track and to read/write the desired record present in the current track.

A study of the above I/O devices reveals that an I/O device basically consists of the following modules :

1. a storage media
2. a read/write head to store/retrieve information from the storage media.
3. a control unit to control operations on the I/O device.

Table 2.1.1 gives a brief decription of some of the I/O devices and their controller's functions.

Usually, the I/O devices are used for mass storage of data and single R/W head is used for storing/retrieving the data. Hence, it is necessary to position the R/W head on the desired data cell of the storage media before reading/writing the data. In order to do so the storage media or R/W head or both, whichever is applicable, must be moved. In electro-mechanical devices (example tape, floppy disk device, card readered) the storage media is usually movable. In these devices, the R/W head may or may not be movable. For example, in the case of card reader and tap device the R/W head is fixed whereas in the case of floppy disk system it is movable. In the case of tape device, the tape (the storage media) is moved to bring the desired data cell under the R/W head whereas in the case of floppy disk system both the R/W head and the floppy disk (the storage media) are moved.

The functionsof the I/O device control unit can be broadly divided into three categories :

(1) to move the R/W head or the storage media or both, whichever is applicable .

(2) to store/retrieve data from the data cell lying under the R/W head

(3) to indicate the current status of the device, if any (for example, in the case of tape device, the status signals are Beginning Of Tape (BOT) and Ending Of Tape (EOT).

The I/O device control unit accepts control commends from the I/O controller and performs the necessary functions as dictated by the commands. The common control signals to the device control unit are,

(1) "MOVE R/W HEAD" or "MOVE STORAGE MEDIA" or "MOVE BOTH R/W HEAD AND STORAGE MEDIA", whichever is applicable.

(2) "READ DATA"

(3) "WRITE DATA"

2.2 FUNCTIONS OF THE I/O DEVICE CONTROLLER :

The functions of the I/O controller can be broadly divided into two categories :

(1) Bringing the desired data cell under the R/W head.

(2) Reading/writing a block of data from the I/O device.

1. Bringing the desired data cell under the R/W head :- Usually the data is recorded on the I/O devices in the form of blocks and the I/O devices are sequential in nature i.e. to access the N^{th} data cell of a block, previous (N-1) data cells must be accessed. Thus in order to read/write a data block, the R/W head must first be positioned at

the beginning of the desired data block before giving read/write control command. Since the "MOVE R/W HEAD" or "MOVE STORAGE MEDIA" or "MOVE BOTH R/W HEAD AND STORAGE MEDIA" control command merely keeps on moving the R/W HEAD/STORAGE MEDIA as long as the control signal is active, the I/O controller must keep track of the amount of movement and deactivate the control signal when the desired data block is under the R/W head. For example, in the case of tape device this operation consists of moving the tape until the desired data block comes under the R/W head and in the case of floppy disk system, this operation consists of positioning the R/W head on a track where the desired data is available.

2. Reading/writing a block of data from the I/O device :-Once the R/W head is positioned at the desired data block, the R/W control command is issued to the I/O device to start reading/writing of the data block. During the R/W operation, the storage media or R/W head or both will be moving continuously and hence the consecutive data cells are read/written as long as the R/W control command is active. The I/O controller must capture the data when it is ready, keep track of the amount of data transferred and deactivate the R/W control command when the desired amount of data has been transferred.

2.3. CLASSIFICATION OF I/O DEVICE INTERFACING SIGNALS :

It is intersting to note that the I/O device interface signals can be classified into four categories :

1. DATA SIGNALS - These signals are used to transfer the data between the I/O device and the I/O controller.

2. CONTROL SIGNALS. - These signals are used to convey the commands from the I/O controller to the I/O device control unit.

3. DEVICE STATUS-SIGNALS - These signals indicate the current status of the I/O device, if any, to the I/O controller.

4. INTERRUPT SIGNALS - These signals are used to interrupt the MPU whenever MPU attention is required by the I/O device.

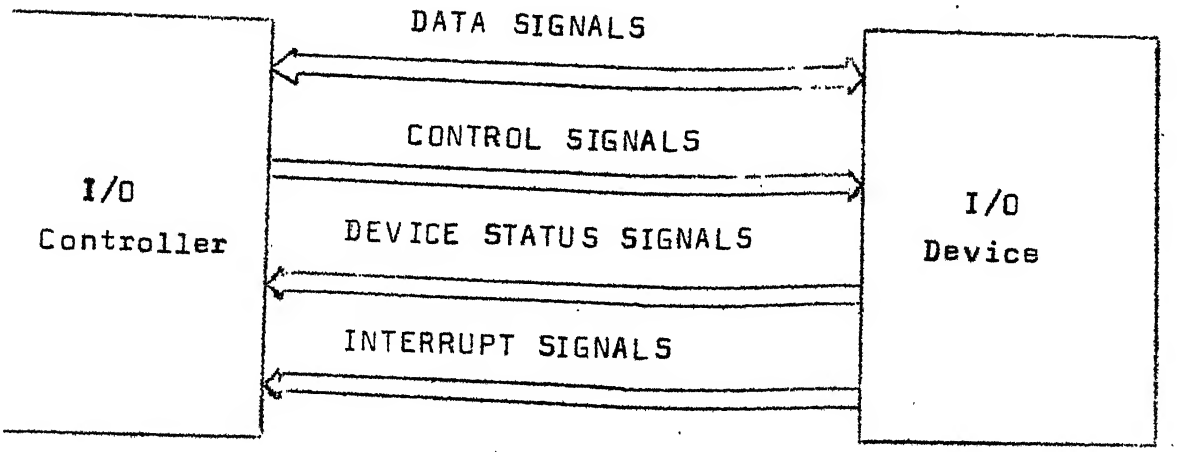


Fig. 2.3.1 : Various Interface Signals of an I/O Device to the Controller.

2.4 GENERALIZED MPU-BASED I/O CONTROLLERS :

Any MPU-based system consists of the following blocks :

1. the basic MPU,
2. the external hardware (conventional electronics), if any,

and 3. the software modules.

Development of generalized MPU-based I/O controller involve making generalization in each of the above blocks. By "generalized controller", it is meant that using the same controller to control different kinds of I/O devices. In the following paragraphs the possible generalizations in each block is discussed.

Microprocessing Unit : The MPU consists of : (1) the microproce processor (2) the system memory, to store data and control programs and (3) interface devices (peripheral components). The first two components are common in any MPU whereas the type and the number of interface devices varies with the system requirements. The requirements of the I/O controllers suggests the generalized MPU shown in Fig. 2.4.1 . It has the following peripheral components :

(1) The hardware timer (the INTEL's programmable interval timer, 8253),

(2) the priority interrupt controller (the INTEL's programable interrupt controller, 8259) and

(3) the peripheral interface devices (the INTEL's programable peripheral interface, 8255). One peripheral interface device is used as data buffer and the other interface device is used to interface control and status signals.

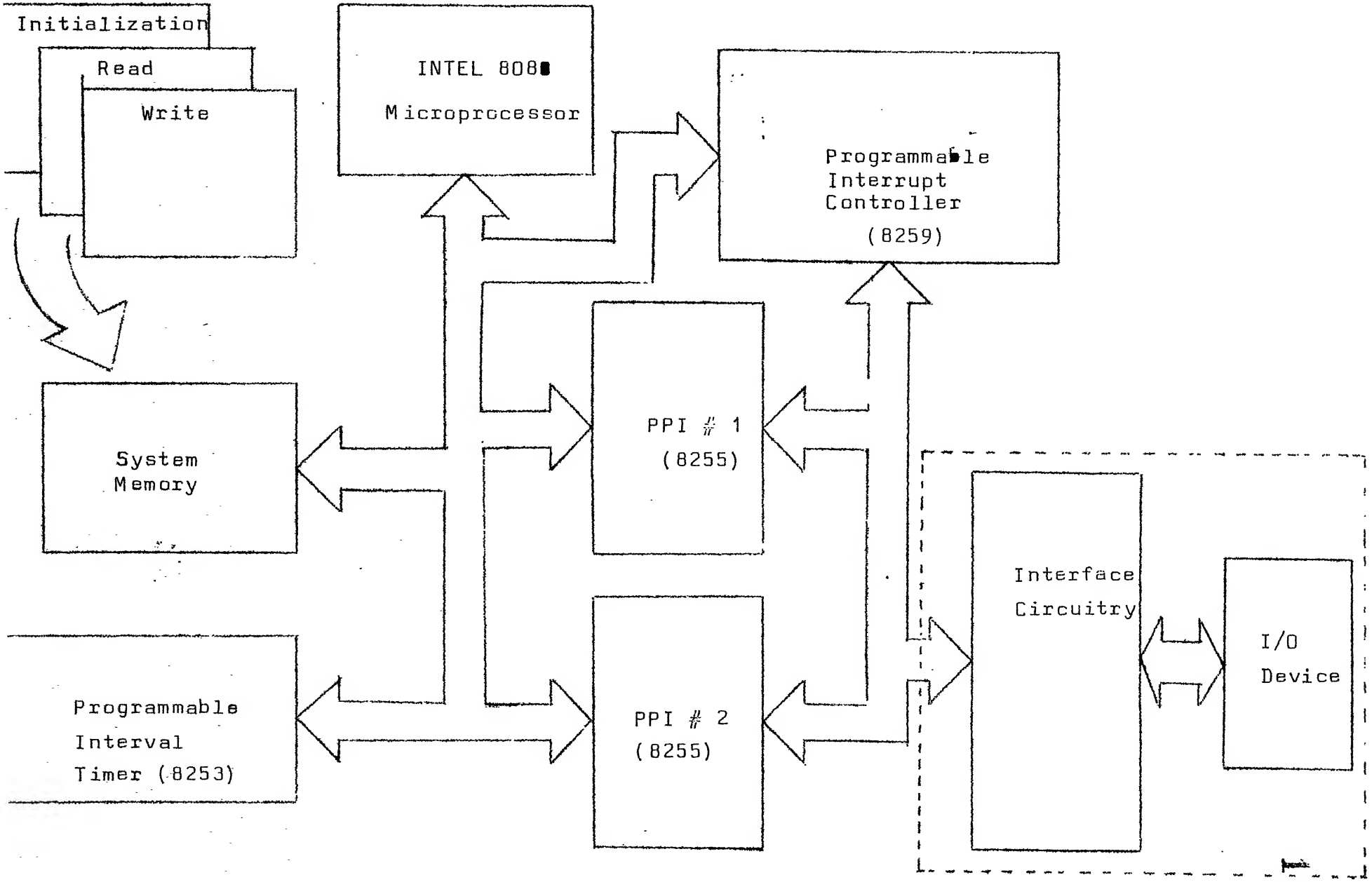


Figure 2.4.1 : A generalized MPU system for I/O Controllers.

More than one peripheral interface devices (8255s) are required if all the control and status signals can not be interfaced with single peripheral interface device. Note that the time delays can be generated and interrupts can be handled with the software also. The necessity and advantages of using the hardware for generating the time delays and handling the interrupts is discussed in detail in the next section with respect to the requirements of I/O controllers. For details on peripheral components refer (5) .

The external hardware : In the case of I/O devices which are used for storing as well as retrieving the data, the data must be retrieved at the same rate as it was recorded. The read clock is used to synchronize the read program to the data rate. Thus before reading the data, it is necessary to adjust the read clock rate to the data rate at which it was recorded. It is assumed that the read clock synchronization to the data rate is achieved by the hardware. The MPU inputs/outputs the data when it is requested by the read/write clock. Handshake mode is assumed during reading/writing the data i.e. data is inputted or outputted when it is requested and an acknowledge signal is generated when the request is serviced. If data is not transferred by the MPU when requested then the hardware generates overflow/under flow error. Apart from the synchronization circuits, the external hardware consists of circuits for pulse shaping the interfacing signals

and other control circuitry whose functions can not be done by the software. It is assumed that the microprocessor is fast enough to handle most of the I/O controller functions and hence the external hardware consists of only the synchronization circuits and pulse shaping circuits. It is obvious that the synchronization circuits and pulse shaping circuits vary with the type of device. However, all the devices of the same kind but having different speeds will have the common synchronization and pulse shaping circuits. For the high speed devices, the external hardware consists of additional hardware which is used to perform some control functions, may be partially, to meet the speed requirements of the devices. Thus absolute generalization in the external hardware is not feasible, it depends on the type and speed of the devices.

Software Modules :

The functions of the MPU and the external hardware have been fixed. The MPU has interface devices to generate time delays, to handle priority interrupts and to interface device signals to the MPU system bus. The external hardware performs the pulse shaping of interface signals and achieves read clock synchronization, if applicable. Now the remaining functions of the I/O controller has to be performed by the software. It has been pointed out that the basic functions of the I/O controller are, (1) to bring the desired data cell

under the R/W head and (2) to transfer desired amount of data to/from the I/O device. These two functions are independent. The second function involves two sub-tasks: 1.to capture the data from the I/O device when it is ready and 2. to process the captured data. The data processing involves keeping track of the amount of data transferred and moving the captured data tothe desired memory location. Each function involves issuing a set of sequence of control commands to the I/O device control unit. Before giving a command, the device check should be made to ensure that the desired operation is carried out at proper stages. Sometimes a time out has to be performed to achieve the desired device status. The typical steps involved, while executing a control function are :

Step 1 : Check the device status. Go to Step 2.

Step 2 : If the status is good to carry out the desired operation then go to Step 3, else go to Step 4.

Step 3 : Give the control command to carry out the desired operation. Go to Step 7.

Step 4 : If a time delay is required to achieve the desired status then go to Step 5 else go to Step 9.

Step 5 : Perform the desired time out. Go to Step 6.

Step 6 : If the status is good to carry out the desired operation then go to Step 3, else go to Step 9.

Step 7 : Wait until the desired operation is completed. Go to Step 8.

Step 8 : If the previous operation is the last operation then go to Step 10, else go to Step 1.

Step 9 : Generate error signal.

Step 10: Stop the execution of the control function.

Apart from the above basic software control modules, the software consists of interrupt service routines, Error processing routines, if any and a routine to initialize the MPU system when power-on. Thus the software modules in an MPU-based I/O controller can be classified as follows:

(1) routines to initialize the MPU system when power-on.

(2) routines to bring the desired data cell under the R/W head.

(3) routines to transfer the desired amount of data to/from the I/O device.

(4) routines to process interrupts.

and (5) routines to process errors, if any.

A "control block" will be maintained in the memory for communication between the executive program and the I/O controller. This block contains the following information:

(1) current operation

(2) starting address of the data

(3) data count (amount of data to be transferred).

(4) amount of data that has been transferred.

and (5) Error status.

Although the various functions of the software have been defined, it is not feasible to generalize the software control modules absolutely i.e. to develop a common software which can be used to handle different kinds of I/O device, This is due to the wide variations in the characteristics of the device from device to device. The control signals, status signals and the interrupt signals change completely from device to device. Thus there is no commonality in the control commands and in their sequence of execution during the execution of a control function. However, the generalization is possible, if the software control is restricted to a particular kind of I/O devices. For example, a generalized software can be developed for all tape devices having different speeds. The variable parameters, which depend on the speed of the tape device, are : 1. start gap time, 2. stop gap time 3. Full tape rewinding time (this time also depends on tape length) 4. BOT and load point gap time (i.e. the time required to reach to the load point from BOT at R/W speed.) 5. Record length time (time required to read a full record, it also depends on record length and type of data transfer i.e. whether it is bit serial data transfer or byte data transfer program). These variables can be passed as parameters to the control modules to make them adaptive to the desired speed requirements. The software can also be generalized to handle both bit serial data transfer (data is transferred one bit

at a time) and byte data transfer (data is transferred 8 bits at a time).

2.5 HARDWARE/SOFTWARE TRADEOFFS:-

Once the system requirements are defined, the designer must almost immediately begin making decisions to establish whether a hardware or a software approach will be used for each sub-task.

The tradeoff possibilities can be explored by considering some of the overall design goals. For example, if a high volume production run is anticipated, a logical goal is to minimize hardware costs since each dollar of cost is amplified by the unit count. In such cases, every attempt should be made to eliminate external control hardware by "loading up" the MPU since control programs in memory are generally more economical than generating the same function with conventional logic circuits.

At the opposite extreme are relatively complex systems (with attendant high engineering development costs) that will be produced in limited quantity. In this case, minimizing development cost may be the more important criteria. In typical MPU systems, the peripherals can be obtained from OEM suppliers with varying amounts of the control and drive circuitry provided. Using peripherals with conventional control and drive electronics will simplify the tasks that must be performed by the MPU and will result

in a shorter, more economical development cycle.

Typical applications fall somewhere between these two extremes. There will usually be three approaches to consider: the older conventional method, the new programmed method and a judicious blend of both.

The primary goal using an MPU based design should be to replace as much hardware as possible with a control program that causes the MPU to duplicate the hardware process. This capability is the primary motivation for using an MPU in the first place. In many cases, this approach can be carried to the extreme of eliminating everything except the family devices (MPU, memory, Interface) and the peripherals themselves. However, in most systems, there are tasks that, if done in hardware, can significantly reduce memory requirements and/or improve the data throughput rate. Depending on the system design goals, it is possible that the dedicated MPU based design can be more expensive than hardwired logic. Of course, the MPU based design is much more flexible and can be programmed to have a higher level of "intelligence" than its hardwired logic counter part. If the system has a potential to grow (long term consideration), or if there is a need to design a flexible system, the MPU based design would be a wise decision. For the above reasons, selection of the configuration should not be based on simplistic hardware/software tradeoffs.

The first task of the system designer is to become familiar with the MPU's characteristics. Knowledge concerning the MPU's ability to handle various aspects of the problem will heavily influence the methods that are finally adopted. Such factors as operating speed, the number of working registers and how they can be used, available control features, I/O techniques, addressing modes, and the instruction set will all influence each stage of the development.

The remainder of this section illustrates by example some of the steps involved in reducing a system specification to individual software and hardware tasks. The various options are discussed in context with the I/O controllers.

2.5.1 MEMORY REFERENCE I/O vs. DMA I/O: -

Memory reference or software I/O refers to the technique of transferring data to and from memory via the MPU and a PPI (programmable peripheral interface). For example, to load memory from a pheriphal device, the data flow would be to (1) PPI to (2) MPU, to (3) memory. This is typically accomplished in software by an "IN" (PPI to MPU) instruction addressing the proper PPI followed by a "STA" (MPU to memory) instruction addressing the desired memory location.

DMA, or direct memory access, is a technique by which pheripheral data transfer to and from memory is

accomplished directly utilizing special DMA hardware. This implies that a DMA transfer is transparent to the MPU, thus costing no software. However, it does affect system operation; the DMA circuitry must disable the MPU and generate the desired data, address and control signals required to transfer data directly to memory. During non-DMA operation, the MPU is driving the system busses and control lines. When DMA is required, the DMA hardware is responsible for driving the busses and some of the control lines, therefore, to avoid contention the DMA must "disable" the MPU.

The low to medium speed I/O devices can be controlled by the microprocessors. The specific Floppy disk system that is chosen as an example, requires a nominal transfer rate of approximately 31.25 K bytes/sec. The INTEL 8080, utilizing the PPI, can accomodate this transfer rate using software techniques. The cassette transfer rate is even slower, approximately 1.5 K bytes/sec. Even though the data block is long (256 bytes), suggesting DMA usage, the byte rate is very slow compared with the MPU's software capability.

2.5.2 SOFTWARE vs. HARDWARE PERIPHERAL SERVICE PRIORTIZING

Either software or hardware techniques can be used to establish the priority by which several interrupts are serviced. In the software approach, the MPU polls the interrupt signals to determine which interrupt signal needs

service. The priority of the interrupt signals with respect to one another is, therefore, determined by the order in which the software performs the poll. Software techniques, in theory, handle any number of interrupts to any sophistication level of prioritizing. In practice, if there are many sources of interrupt requests, the time required to find the appropriate interrupt can exceed the time available to do so. In this situation, external prioritizing hardware can be used to speed up the operation. The INTEL's programmable interrupt controller (8259) is included in the MPU system to handle the I/O devices which require the external prioritizing hardware.

The INTEL's peripheral component, programmable interrupt controller (8259), functions as an overall manager in an interrupt-driven system environment. It accepts requests from the peripheral equipment, determines which of the incoming requests is of the highest importance (priority), ascertains whether the incoming request has a higher priority value than the level currently being serviced and issues an interrupt to the CPU based on this determination. The 8259 is programmed by the system's software as an I/O peripheral. A selection of priority modes is available to the programmer so that the manner in which the requests are processed by the 8259 can be configured to match the system requirements. The priority modes can be changed or reconfigured

dynamically at any time during the main program. This means that the complete interrupt structure can be defined as required, based on the total system environment.

In the case of tape casette (when data is transferred in bit serial form) and floppy disk system, the software prioritizing methods are partially used. Some interrupts require service as quickly as possible once it occurs. In such a case, the MPU can ignore all other service requests and go into a programmed waiting loop until the device asks for service. In the case of above mentioned devices, when the device is transmitting data to the MPU, the MPU repeatedly asks the device if data is coming and responds immediately to a request for service.

2.5.3 SOFTWARE VS HARDWARE TIMER

In most systems, the MPU must occasionally perform a time out requirement before continuing with the program. Software can be used to generate the delay in a straightforward fashion by loading a register and decrementing it to zero and then allowing the program to continue. The amount of delay is determined by the value loaded into the register.

The hardware approach is useful in two situations: 1. When another useful task can be performed by the MPU while a time out is taking place; 2. When two or more simultaneous but independent delays or time outs are required for proper system operation.

The INTEL's Programmable Interval Timer (8253) is incorporated for both purposes. It is used as a matter of efficiency by several of the peripheral subroutines and as a matter of necessity by some I/O devices.

2.6 SOME GUIDE LINES TO BE FOLLOWED WHILE INTERFACING THE DEVICE SIGNALS TO THE MPU SYSTEM :

The device signals are interfaced to the MPU system data bus through the Programmable Peripheral Interface (PPI, 8255). The IN/OUT instruction transfers 8-bit data to/from a port of a PPI. A device interface signal can be interfaced to the system bus through any of the available I/O data line of PPI. But it is advantageous, if all the signals belonging to the same class are interfaced through a single port. This will enable us to access all the signals belonging to the same class through a single INPUT/OUTPUT instruction. The following example clearly illustrates the advantages.

The following is the set of control signals in a tape device.

RD- WRT - Read = 1, write = 0

Erase - Rd Enable - Erase = 1, Rd Enable = 0

CRC Reset - CRC Enable - Reset = 1, Enable = 0

Rd/wrt Speed - Search Speed - Rd/wrt speed = 0, Search Speed =

Fwd direction- Reverse Direction - Fwd = 1, Rev = 0

CRC shift - wrt data enable - CRC shift = 0, wrt data Enable = 1.

Stop - Go - Stop = 1, Go = 0 .

There are seven control signals and these can be interfaced through a single Port. The signals are interfaced through PORTA of 8255 as shown below. With this technique, we can have various control words, as shown in figure 2.6.2 .

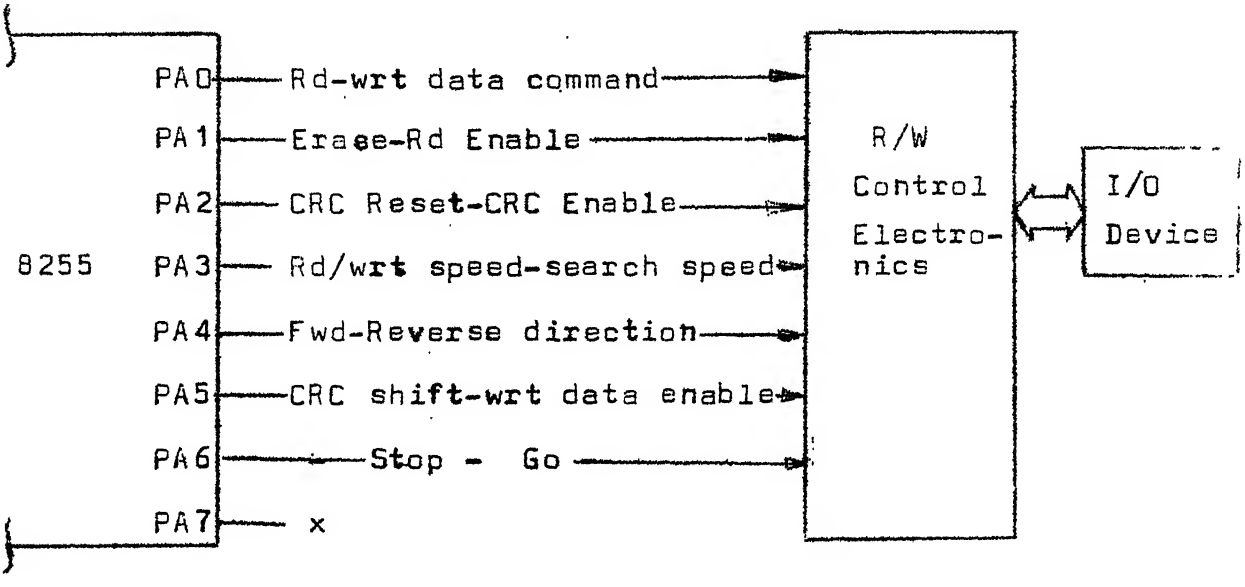


Figure 2.6.1

corresponding to the various operations on the I/O device. The following is the list of possible control words in the case of Tape device. With the above technique, any control commend can be communicated to the I/O device with single OUTPUT instruction.

Port A	Control Word								OPERATION
	7	6	5	4	3	2	1	0	
	X	0	0	0	0	1	1	1	Rewind tape at search speed.
	X	1	1	1	0	1	1	1	Stop tape in Read-Forward and Erase Mode.
	X	0	0	1	0	1	1	1	Move Fwd at RD-WRT speed in Erase mode with Read electronics Enabled.
	X	0	1	1	0	1	1	0	Move forward at RD-WRT speed in Erase mode with write electronics Enabled
	X	0	0	1	0	1	0	1	Move forward at RD-WRT speed with Rd-Enabled, CRC generator Reset, write data enabled and Read electronics enabled
	X	0	1	1	0	0	0	1	Move Forward at Rd-wrt speed with Rd Enable, CRC generator enabled.
	X	0	0	1	0	1	1	1	Move fwd at RD-WRT speed in Erase mode with Rd electronics enabled and CRC shift enabled.
	X	0	1	1	0	1	0	0	Move Fwd at RD-WRT speed in Rd Enabled mode with CRC reset, write data enabled.
	X	0	1	1	0	0	0	0	Write-Forward - CRC Enable Control word .

Fig. 2.6.2

The steps that are involved in developing a generalized microprocessor based I/O controllers are :

Step 1 : Study the device operating characteristics and speed requirements.

Step 2 : Identify the status, control and interrupt signals.

Step 3 : Develop the generalized MPU system shown in Figure 2.4.1.

Step 4 : Interface the data signals to the MPU system through PPI # 1 and the status and control signals through PPI # 2 following the guidelines given in Section 2.6.

Step 5 : Find the priorities of interrupt requests, if any, and handle them with the Programmable Interrupt Controller (8259).

Step 6 : Develop the pulse shaping circuits and synchronization circuits and Error detect circuits.

Step 7 : Develop the following software modules :

(i) a routine to initialize the MPU system when power-up.

(ii) routines to bring the desired data cell under the R/W head.

(iii) routines to READ/WRITE a data record and

(iv) routines to process Errors, if any.

Step 8 : Determine the device speed dependent variables and generalize the software modules by passing the values of these variables as input parameters.

Step 9 : Evaluate the software control routines and find whether they are meeting the device speed requirements.

Step 10 : If the speed requirements are not met then modify the software modules and/or replace some software with hardware and go to Step 9.

CHAPTER 3

DEVELOPMENT OF MPU BASED I/O CONTROLLERS

In this chapter MPU based I/O controllers are developed for the following devices following the methodology described in the last chapter :

1. Printer
2. Tape cassette system
3. Floppy disk system

All the software routines are explained with the aid of a language construct similar to the programming language, PASCAL, in Tables 3.1.1 through 3.3.4 .

3.1 Printer Controller :

A SEIKO AN 101F printer is selected as an example. To determine the printer controller functions, the operating characteristics are studied first.

3.1.2 Operating Characteristics of the Printer :

The SEIKO AN- 101F printer employs a continually rotating print drum mechanism using what is referred to as the flying printer technique. The printing principle of the mechanism is indicated schematically in Figure 3.1.1 .

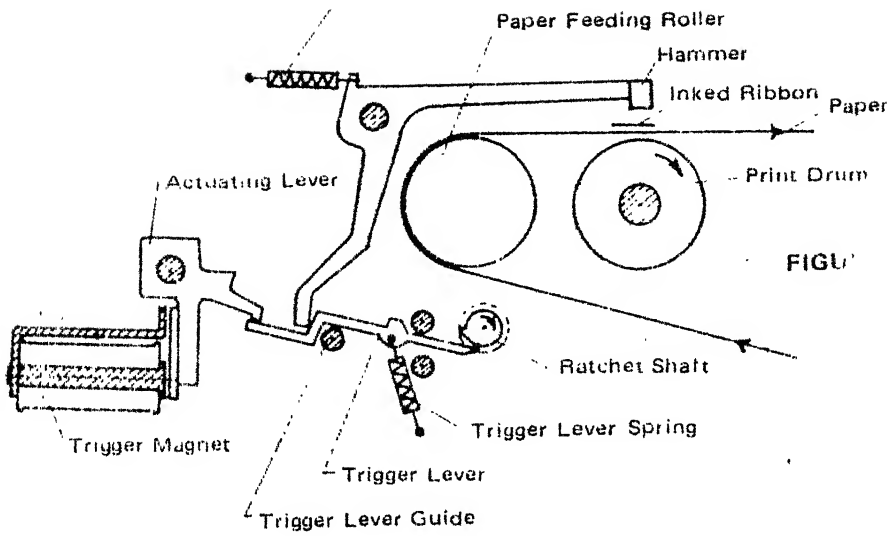


Figure 3.1.1 : SEIKO AN -101F Printing Mechanism

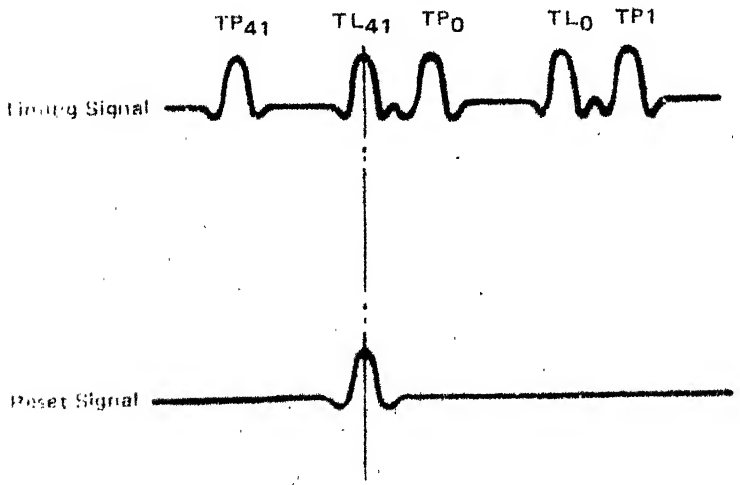


Figure 3.1.2 : Timing Signals

The printer drum and the ratchet shaft are geared together and rotate continuously in the direction shown. During a non-print condition, the right end of the trigger lever is removed from the ratchet's pawl locus by the downward force of the trigger lever spring. In the non-printing condition, the trigger magnet is not actuated and the hammers are lifted upward to a neutral position by the hammer lever springs.

When actuated, the trigger magnet's actuating lever forces the opposite end of the trigger lever into the locus of the ratchet pawl. During its next rotation, the pawl will engage the right end of the trigger lever causing a downward motion to the right hand end of the hammer. The hammer thus strikes through the inked ribbon and paper, causing the character then under the hammer to be printed.

Any of 42 characters (alphanumeric plus special characters x, , , , - , . and /) may be printed in a 21-column format. Each position has a complete character set spaced evenly around the drum. Because of a 42:1 gear ratio, the ratchet rotates 42 times for each complete drum rotation. Hence each character of the set is positioned under a print hammer once during every rotation of the drum.

From this brief description of the printer mechanisms characteristics, it is evident that the basic function of the I/O controller is to actuate the hammers at just the right time if printing is to occur. Timing signals are generated electromagnetically by means of detection heads and ferrite magnets associated with the ratchet shaft and drum. Rotation of the ratchet shaft generates signals TP and TL for each of the 42 characters. TP provides timing for energizing the trigger magnets, TL for de-energizing. A reset signal R is generated by each complete rotation of the drum. The timing signals are shown in figure 3.1.2 .

Each hammer driver is controlled by one of the PPI's (8255#1) 21 data lines. When the STROBE control signal is made active, all the hammers whose corresponding PPI data lines are "1" are activated. The PPI#2 is used to interface control and status signals.

The Control signals are :

PPI#2 PC0 STROBE

PC1 PAPER/RIBBON FEED

The status signals is :

PPI#2 PC4 TP/TL timing signal

The interface of the above signals is shown in Figure 3.1.3 .

The programmable interrupt controller (8259) is

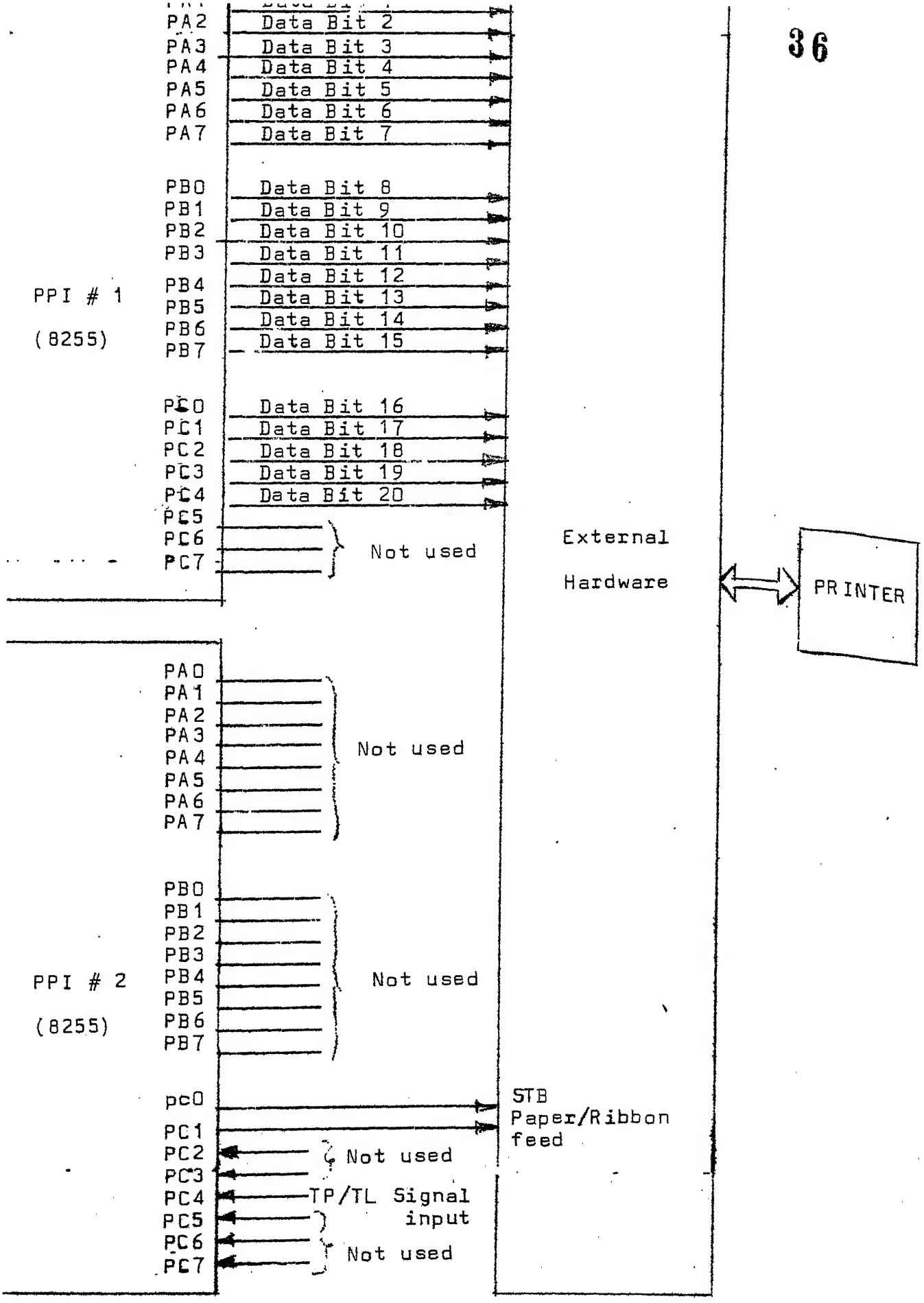


Figure 3.1.3 : Printer Functional Interface.

used to handle interrupt signals. The interrupt signals are :

IR0 Interval Timer (8253) interrupt

IR 1 TP/TL timing signal interrupt

IR2 Reset signal 'R' interrupt.

External Hardware : Since the printer is an output device, read clock synchronization circuits are not required. Each PPI data line is ANDed with the STROBE so that the hammers are actuated when strobed. Apart from theee AND gates the hardware consists of pulse shaping circuits.

SOFTWARE DESCRIPTION :

The basic task, or algorithm , of the control program is to examine the text of the message to be printed and make sure that the appropriate bits in the PPI#1's output ports, PORTA, PORTB and PORT C,are set at the proper time.

Understandng of the operation is aided by regarding the time for one print drum rotation as forty-two equal intervals, t_0 through t_{41}. With this is mind, note that all similar characters in the text are printed simultaneously, i.e. all 0's are printed during t_0, all 1's during t_1 etc. For example, if the text requires the letter C in coulumns 3 and 9, column hammers 3 and 9 must be engaged during the time interval t_{12} during which all C's are under the hammers.

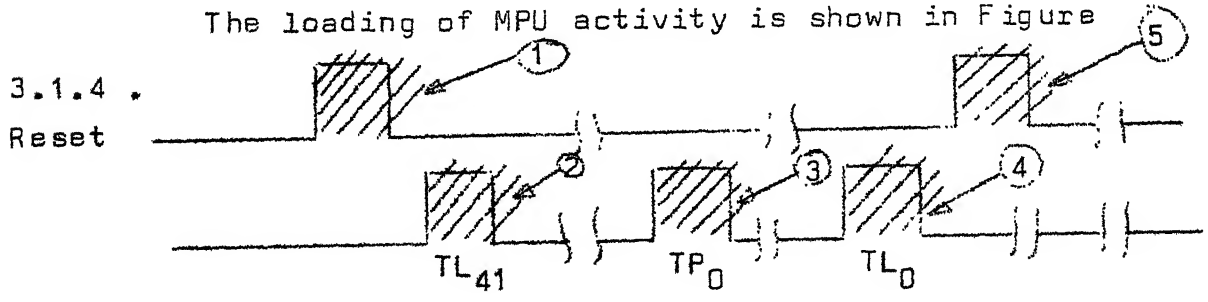
Following each "TL" interrupt, the MPU examines the the entire message to see if there are any characters to be

printed during the next time interval . Messages are stored in memory in 21- byte block with each memory position corresponding to a printer column position.

A 42-byte character file corresponding to the printer's character set is stored in a 42 byte memory block in the same sequence as it appears on the printer drum. As each TL interrupt is serviced, the character file pointer is incremented pronting to the address of the next character on the drum.

The MPU then compares every character of the text to the current character file character, keeping a running column counter as it does so. Each data line of the PPI#1 is set or cleared depending on whether or not the respective text characters matched the character file character .

The loading of MPU activity is shown in Figure 3.1.4 .



- Loops until TL_{41} flag is set.
- Selects hammers to be engaged at TP_0, and enables interrupt TP/TL.
- Engages selected hammers.
- Disengages hammers and selects hammers to be engaged at next TP.
- Terminating the print cycle ; then performs a paper/ribbon feed.

Figure 3.1.4 Printer Loading of MPU Activity.

The control program of PRINTER CONTROLLER is broken into the following sub tasks :

1. INITIALIZATION : This routine is used to initialize the peripheral interface devices during system power-up. A detailed explanation of this routine is given in Table 3.1.1 .
2. PRINTER ENABLE : This routine is called whenever a line of text is to be printed. This routine initializes all the variables and enables Reset signal interrupt to initiate the printing at the begining of new print drum cycle. A detailed explanation of this routine is given in Table 3.1.2 .
3. RESET SERVICE ROUTINE : This routine services the reset signal, R, interrupt. A detailed explanation of this routine is given in Table 3.1.3 .
4. PRINT SERVICE ROUTINE : This routine services the timing signal, TL/TP, interrupt. A detailed explanation of this routine is given in Table 3.1.4 .
5. Input text scan routine, "SCAN" : This routine scans the input text and selects the hammers which are to be engaged at next TP pulse by setting the appropriate data lines of PPI#1. A detailed explanation of this routine is given in Table 3.1.5 .

```
procedure INITIALIZE;
   begin
        set PORT A,PORT B and PORT C(LOWER)
        of PPI#1 to output mode;
        reset STROBE flag;
        reset PAPER/RIBBON FEED flag;
        initialize PROGRAMMABLE INTERVAL TIMER;
        initialize PROGRAMMABLE INTERRUPT CONTROLLER;
        mask all INTERRUPT REQUESTS
   end;
```

Table 3.1.1 INITIALIZATION Routine of PRINTER.

```
procedure PRINTENABLE;
      begin
           reset PRINTDONE flag;
           initialize CHARACTER FILE POINTER;
           initialize CHARACTER COUNT variable;
           set BEGINING OF PRINT DRUM CYCLE flag;
           enable RESET INTERRUPT REQUEST
      end;
```

Table 3.1.2 PRINTER ENABLE Routine of PRINTER.

```
procedure RESETPROCESS;
      procedure TIMERPROCESS;
      /*This is the INTERVAL TIMER interrupt (IRO) service*/
     /*routine.This routine is executed when the PAPER FEED*/
     /*TIME elapses.*/
           begin
                reset PAPER/RIBBON FEED flag;/*stop paper feed*/
                mask all INTERRUPT REQUESTS;
                set PRINT DONE flag
           end;
```

contd. Table 3.1.3

```
begin/*RESETPROCESS*/
   if BEGINING OF PRINT DRUM CYCLE then
      begin
         loop until TP/TL SIGNAL flag is set;
      call SCAN;
      enable TP/TL SIGNAL INTERRUPT REQUEST;
      reset BEGINING OF PRINT DRUM CYCLE flag
      end
    else
         begin
      loop until TP/TL SIGNAL flag is set;
      reset STROBE flag;
      set PAPER/RIBBON FEED flag;
      set PROGRAMMABLE INTERVAL TIMER to
      PAPER FEED TIME;
   end
end;
```

Table 3.1.3 RESET SERVICE Routine of PRINTER.

```
procedure TIMINGPROCESS;
   begin
         if TL SIGNAL INTERRUPT then
            begin
               reset STROBE flag;/*Disensase hammers*/
               call SCAN;/*Select hammers to be ensased */
               /*at next TP signal*/
            end
         else set STROBE flag;/*Ensase hammers*/
   end;
```

Table 3.1.4 PRINT SERVICE Routine of PRINTER.

```
procedure SCAN;
     begin
           initialize INPUT TEXT POINTER to zero;
           repeat
           if CF CHARACTER = TXT CHARACTER then
                  set CARRY
           else clear CARRY;
           if PORT A to be updated then
                  shift CARRY to PORT A
           else if PORT B to be updated then
                  shift CARRY to PORT B
                else update PORT C;
           increment INPUT TEXT POINTER;
           until INPUT TEXT POINTER > 20;/*21 COLUMN format*/
           increment CHARACTER FILE POINTER
    end;
```

Table 3.1.5 SCAN Routine of PRINTER.

3.2 TAPE CONTROLLER:

Data is recorded on the tape in the form of blocks with each data block consisting of : (A) preamble (1 byte (B) data (4 to 256 bytes) including the Cyclic Redundancy Check Character (2 bytes) and (C) a postamble (1 byte). The data recording format is shown in Figure 3.2.1.

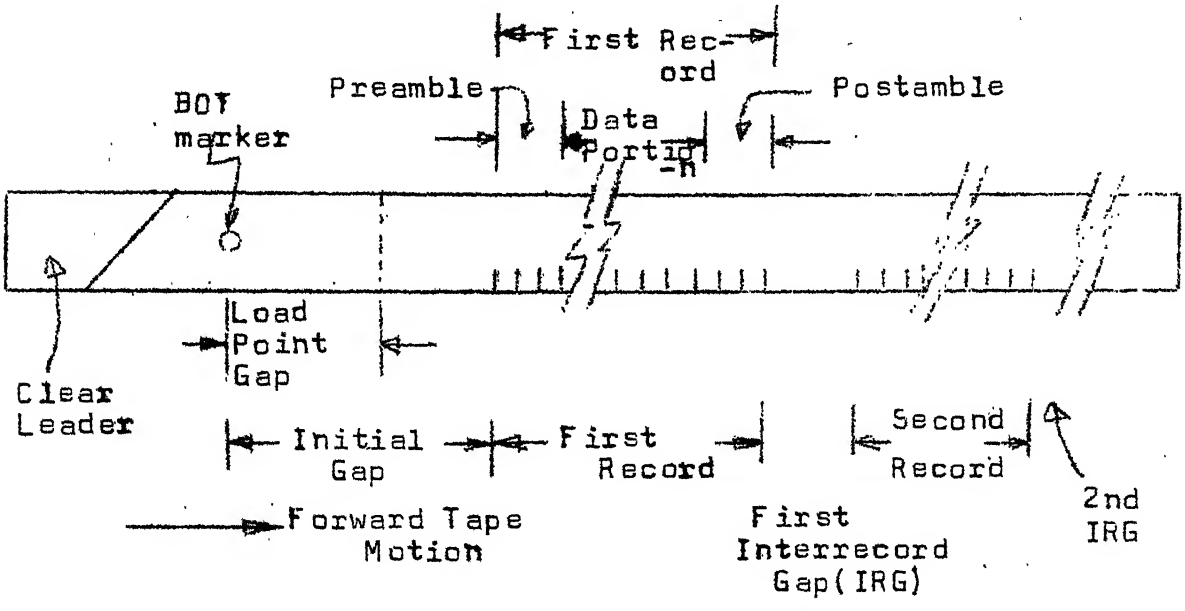


Figure 3.2.1 . The Data Recording Format on the Tape.

3.2.1 Operating Characteristics of the Tape Device : Three control lines are provided for control of tape motion: 1. STOP/GO 2. FORWARD/REVERSE 3. SEARCH/REWIND or READ/WRITE SPEED.

The tape can be moved either in forward or reverse direction. When the data is to be transferred the tape is moved at READ/WRITE speed in the forward

direction and when the tape is to be rewound or when a given record is to be brought under the R/W head then the tape is moved at SEARCH/REWIND speed.

After giving the motion command, the tape motion begins after certain delay and reaches stable speed after some more delay. This is the startgap delay and the data should be transferred only after the startgap delay i.e. when the tape reaches its full speed.

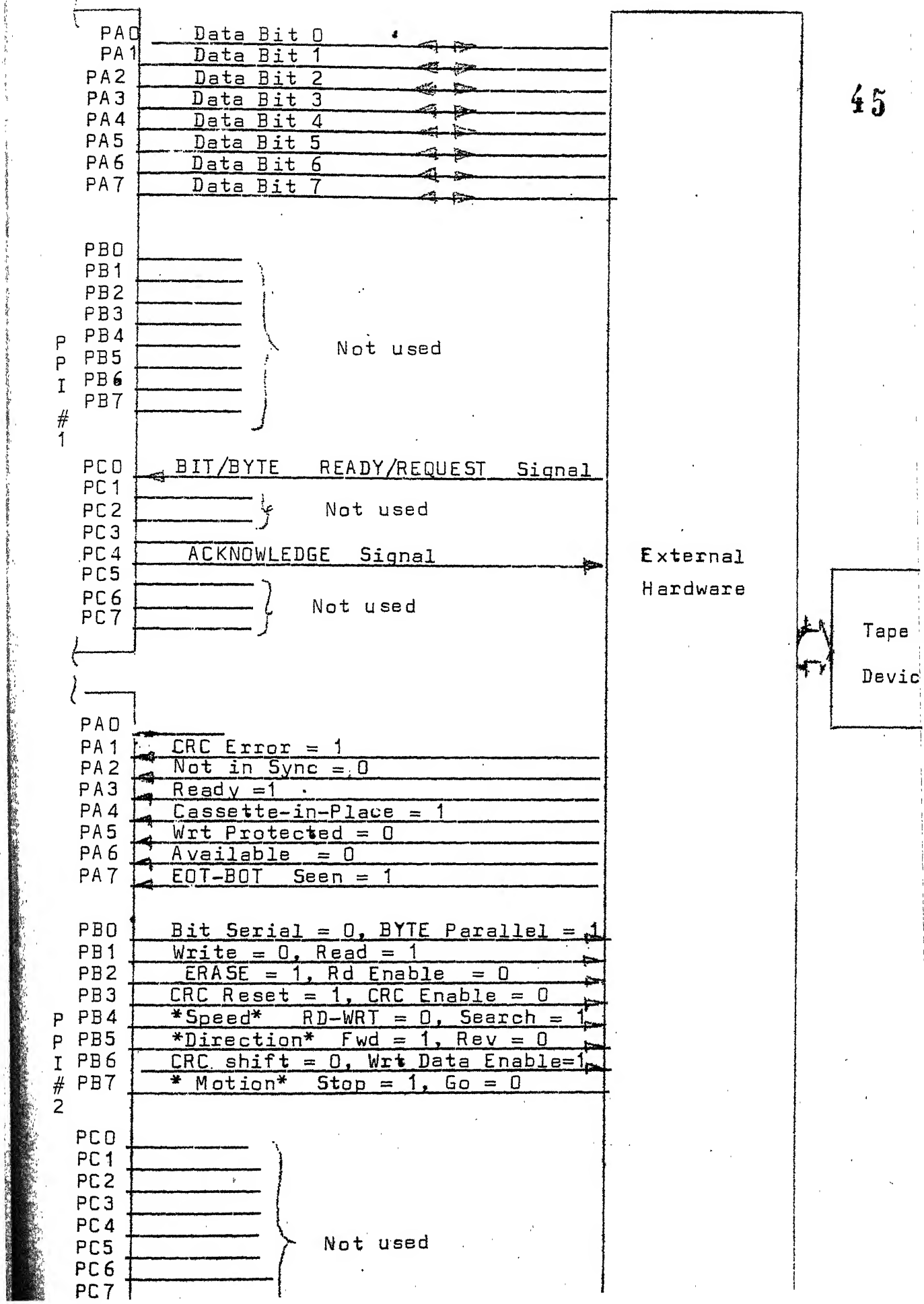
Photo-detectors are used for sensing End of Tape (EOT) and Beginning of Tape (BOT). The device is provided with both a cassette-in-place sensor and a file-protect sensor.

In a typical tape system, many functions must be performed, however, only the following basic routines are described.

1. REWIND TO LOAD POINT (Load Point is the logical beginning of the tape).
2. READ A RECORD and
3. WRITE A RECORD

The control and status signals are interface to the MPU system bus through PPI#2 as shown in Figure 3.2.2 . The control signals are :

PPI # 2	PB1	Write = 0, Read = 1
	PB2	Erase = 1, Rd Enable = 0
	PB3	CRC Reset = 1 , CRC Enable = 0


PPI #1
PA0 Data Bit 0
PA1 Data Bit 1
PA2 Data Bit 2
PA3 Data Bit 3
PA4 Data Bit 4
PA5 Data Bit 5
PA6 Data Bit 6
PA7 Data Bit 7
PB0
PB1
PB2
PB3
PB4
PB5
PB6
PB7
Not used
PC0 BIT/BYTE READY/REQUEST Signal
PC1
PC2
Not used
PC3
PC4 ACKNOWLEDGE Signal
PC5
PC6
PC7
Not used
External Hardware
Tape Devic
PPI #2
PA0
PA1 CRC Error = 1
PA2 Not in Sync = 0
PA3 Ready = 1
PA4 Cassette-in-Place = 1
PA5 Wrt Protected = 0
PA6 Available = 0
PA7 EOT-BOT Seen = 1
PB0 Bit Serial = 0, BYTE Parallel = 1
PB1 Write = 0, Read = 1
PB2 ERASE = 1, Rd Enable = 0
PB3 CRC Reset = 1, CRC Enable = 0
PB4 *Speed* RD-WRT = 0, Search = 1
PB5 *Direction* Fwd = 1, Rev = 0
PB6 CRC shift = 0, Wrt Data Enable=1
PB7 * Motion* Stop = 1, Go = 0
PC0
PC1
PC2
PC3
PC4
PC5
PC6
PC7
Not used

PB4 * Speed* RD-WRT = 0, Search = 1

PB5 * Direction * Fwd = 1, Rev = 0

PB6 CRC shift = 0, Wrt Data Enable = 1

PB7 *Motion * Stop = 1 , GO = 0 .

The status signals are :

PPI#2 PA1 CRC Error = 1

PA2 Not in Sync = 0

PA3 Ready = 1

PA4 Cassette-in-Place = 1

PA5 Wrt protected = 0

PA6 Available = 0

PA7 EOT-BOT Seen = 1 .

The interrupt signals are interfaced through the programmable interrupt controller (8259). The interrupt signals are :

IR0 Interval Timer (8253) interrupt

IR1 EOT-BOT time out interrupt

IR2 Overflow-underflow interrupt

3.2.2 Enternal Hardware Description : The data to be recorded on the tape is presented to the tape transport in Non-Return-to-zero (NRZ) format but is recorded in phase encoded (PE) format. The data conversion is performed by the hardware. The write data (or CRC data) is gated through the NRZ to PE data converter before recording on the tape. The write data is also sent through a Cyclic Redundancy Check Character (CRCC). The CRCC is

appended to the data block and the CRC data passes through the same circuitry as the write data for conversion to the P.E. format for recording. Both the preamble and the postamble are 8-bit patterns of alternating ones and zeros (01010101-MSB). (This can be used to establish the data rate during data recovery since there is a single transition per bit). During the write operation, the CRC generator is enabled (by clearing the data line PB3 of PPI# 2) after the preamble data has been written. The CRC generator remains enabled throughout the data block. At the end of the data block, the CRC data is shifted out of the generator into the Write circuitry (clearing PB6 of PPI#2 shifts CRC data into write circuitry whereas setting the PB6 data line moves the write data from PPI#1 to the write circuitry).

During a Read operation (PB1 of PPI#2 is set) the write circuits are disabled and the read head passes the read signals onto theread circuits which amplify and convert the read signals to logic levels in P.E. format. The P.E. read data goes to the phace locked loop data recovery circuit which decodes the data and clock signals. The P.E. data also goes to a monostable multivibrator which is used to detect gaps during a search operation. The Read data goes to the

PPI # 1 directly (PORTA or PORT B) while the recovered clock goes to the PPI # 2 (PC0) via the clock selector circuit.

During a Read operation, the CRC Generator is turned on after the preamble has been read and remains on throughout the data block, including the appended CRC character. At the end of the CRC character the CRC Error line is examined to know whether the data has been read correctly.

An underflow-Overflow interrupt will be generated by the hardware to abort the operation in the event of such an error.

3.2.3 SOFTWARE DESCRIPTION : The basic tasks of the controller are : 1. INITIALIZATION. 2. Rewind the tape to the load point, 3. Read a variable length record and 4. Write a variable length record.

1. INITIALIZATION : This routine is used to initialize the peripheral interface devices during system power-up. A detailed explanation of this routine is given in Table 3.2.1.

2. Rewind to Load Point routine : When a tape cassette is inserted in the Drive, it may be at clear Leader either on the BOT or EOT end, or it may be in the "Middle" of the tape between BOT and EOT markers. A number of different schemes may be

used to move the tape to the Load Point. The method used may be either completely automatic or require some operator intervention. The Rewind to Load Point operation described here assumes that the tape has, at some prior time, been advanced past the BOT marker and it is desired to rewind the tape to the Load Point. A detailed explanation of this routine is given in Table 3.2.2.

3. Read routine : The READ PROCESS procedure checks the tape status and brings the tape up to speed if the status is good. The tape status check consists of checking for Tape Available, Ready, Cassette in Place, In Sync, EOT seen and CRC Error. Whenever the tape is stopped, the hardware sets the In Sync and CRCC Error status bits to a good status. This allows a single Read status check subroutine, TPRDSTATUS, to be used both while the tape is stopped and while it is in motion. The reading of the data will be started when the Interval timer (8253) interrupts the MPU after the start gap delay.

There are three possible sources of Interrupts during the execution of the READ program. They are 1. Overflow Interrupt 2. EOT interrupt and 3. Timer interrupt. The overflow

interrupt occurs if the MPU does not read the next Data Bit/Byte when its presence is indicated by the Bit/Byte Ready signal. The operation will then be aborted by the Overflow Interrupt. The EOT interrupt should not normally occur during the Read operation since the EOT single-shot period is set to a time greater than the length of one record. This implies that even if an EOT transition is seen on starting Read motion, there is enough time to complete that record before being interrupted by the EOT single-shot.

A detailed explanation of the Read routine is given in Table 3.2.3.

4. Write Routine : The Write Record Procedure checks the tape status and brings the tape up to speed if the status is good. The writing of one complete record (Preamble, Data, CRCC and Postamble) begins after the stop gap delay. If more than one record is to be written , tape motion is not stopped in Inter record gaps. The Write Word Procedure writes a byte on to the tape in bit serial form.

There are three possible sources of interrupts during the execution of the Write Program. They are 1. Underflow Interrupt, 2. EOT Interrupt

and 3. Timer Interrupt. The Underflow Interrupt occurs if the MPU does not provide the next Data Bit/Byte when it is requested by the Bit/Byte request signal. The operation will then be aborted by the MPU. The EOT interrupt should not normally occur during the Write operation since the EOT single-shot period is set to a time greater than the length of one record. If no Bit/Byte request is seen, then the write operation is aborted after a time slightly longer than the length of one record. The Interval timer is set to a time corresponding to the length of one record before the beginning of writing of a Data Record. A detailed explanation of the Write routine is given in Table 3.2.4.

The software is generalized by passing the device speed dependent parameters to the control routines. These parameters are STARTGAPTIME, STOPGAP-TIME, RECDLENGTHTIME (the time required to read/write one full record), REWINDTAPETIME (the time required to rewind full tape) , and BOT$LDPT$GAPTIME (the time required to move the tape from BOT to the Load Point at READ/WRITE Speed). The Read and Write Routines are further generalized such that they can transfer the data both in Bit Serial Form and in Byte parallel form. The MPU can transfer the data

```
rocedure INITIALIZE;
   besin
       set PORT A and PORT C(LOWER) of PPI#1 to input mode;
   set PORT B and PORT C(UPPER) of PPI#1 to output mode;
   set PORT A of PPI#2 to input mode;
   set PORT B and PORT C of PPI#2 to output mode;
   reset ACKNOWLEDGE flas;
   initialize PROGRAMMABLE INTERVAL TIMER;
   initialize PROGRAMMABLE INTERRUPT CONTROLLER;
   mask all INTERRUPT REQUESTS
  end;
```

able 3.2.1 INITIALIZATION Routine of TAPE DEVICE.

```
rocedure RELPPROCESS;
     function TPRELPSTATUS;/*Checks whether tape status is */
     /*sood to carry out REWIND TO LOAD POINT operation.*/
         besin
             input TAPESTATUS;
             mask out unwanted bits;
        test AVAILABLE and READY flass;
        return TAPESTATUS
         end;
     rocedure BOTEOTPROCESS;
     /*This routine is executed when BOT-EOT TIME OUT signal */
     /*interrupts the MPU durins Rewind to BOT operation*/
     besin
            output MOVE TAPE AT RD-WRT SPEED control word;
            save STACK POINTER;
            set MOVING FORWARD TO BOT OP flas;
            set PROGRAMMABLE INTERVAL TIMER to 1000
            milliseconds ;/*To abort operation if BOT is */
            /*not sensed soon.*/
            loop until BOT is sensed;
            set PROGRAMMABLE INTERVAL TIMER to
            BOT-LOAD POINT GAP TIME;/*To stop tape when it*/
            /*reaches Load Point*/
    end;
```

contd. Table 3.2.2

```
    procedure TIMERPROCESS1;/*This routine is executed when */
    /*INTERVAL TIMER interrupts the MPU durins REWIND TO BOT*/
    /*operation*/
          begin
              output STOP TAPE control word;
              set REWIND TO BOT OPERATION ABORTED OP flas
     end;
    procedure TIMERPROCESS2;
  /*This routine is executed when INTERVAL TIMER */
    /*interrupts the MPU when tape is movins to BOT marker*/
        begin
            set MOVE FORWARD TO BOT OPERATION ABORTED OP flas;
            restore STACK POINTER;
            output STOP TAPE control word
        end;
   procedure TIMERPROCESS3;
   /*This routine is executed when INTERVAL TIMER interrupts*/
  /*the MPU when tape is movins forward to Load Point*/
       begin
           check TAPE STATUS;
           if TAPE STATUS is sood then
               set REWIND TO LOAD POINT OPERATION COMPLETED
                OP flas
           else
               set MOVE FORWARD TO LOAD POINT OPERATION ABORTED
                OP flas;
           output STOP TAPE control word
     end;
    begin/*RELPPROCESS*/
        output REWIND TAPE AT SEARCH SPEED control word;
        set REWINDING OF TAPE IN PROGRESS OP flas;
        if TPRELPSTATUS is sood then
           begin
               set PROGRAMMABLE INTERVAL TIMER to REWIND TAPE
                TIME ;/*To abort operation if BOT is not sensed*/
                      /*soon*/
           end
        else
            begin
                output STOP TAPE control word;
                set REWIND TO BOT OPERATION ABORTED OP flas;
            end
    end;
```

Table 3.2.2 REWIND TO LOAD POINT Routine of TAPE DEVICE.

```
procedure READPROCESS(WORDCOUNT);
    function TPRDSTATUS;
       /*This routine checks whether the TAPE STATUS is good */
      /*to carry out the READ operation*/
      begin
          input TAPE STATUS from PORT A of PPI#2;
          mask out unwanted bits;
          test AVAILABLE,CASSETTE-IN-PLACE,READY,INSYNC
            and CRC ERROR;
          return TAPE STATUS
      end;
    procedure TIMERPROCESS1;/*This routine is executed when */
    /*the MPU is interrupted by the INTERVAL TIMER when tape*/
    /*is moving in START GAP*/
         procedure TIMERPROCESS2;/*This routine is executed */
         /*when the MPU is interrupted by the INTERVAL TIMER*/
         /*when the time required to read a record elapses*/
             begin
                 set READ ERROR INCOMLETE OP flag;
                 output STOP TAPE control word;
                 restore STACK POINTER
             end;
        procedure BOTEOTPROCESS;
        /*This routine is executed when the MPU is interrupted*/
       /*by the BOT-EOT TIMEOUT signal*/
           begin
               set EOT SEEN OP flag;
               output STOP TAPE control word;
           end;
        procedure OVUNDFLOWPROCESS;/*This procedure is executed*/
         /*when the MPU is interrupted by the OVERFLOW INTERR-*/
         /*UPT signal*/
           begin
               output STOP TAPE control word;
               set OVER FLOW OP flag
           end;
        begin/*TIMERPROCESS1*/
            clear BIT/BYTE READY flag;
            set PROGRAMMABLE INTERVAL TIMER to RECORD LENGTH
             TIME;/* To abort operation if reading of the */
               /*record is not completed within RECORD*/
                /*LENGTH TIME*/
            enable READ MODE;
            set BIT COUNTER to 8;
             read PREAMBLE BYTE;
            enable CRC GENERATOR and OVERFLOW INTERRUPTS;
```

contd. Table 3.2.3

```
        if BIT SERIAL DATA TRANSFER then
           repeat
               set BIT COUNTER to 8;
               repeat
                   loop until BIT/BYTE READY flag is set;
                   shift the READ BIT into the DATA BUFFER;
                   update BIT COUNTER;
                   clear BIT/BYTE READY flag;
               until BIT COUNTER=0;/*WORD assembled*/
               update WORD COUNTER;
               update DATA BUFFER POINTER
         until WORD COUNTER=0/*RECORD assembled*/
        else
           /*BYTE TRANSFER*/
                repeat
                    loop until BIT/BYTE READY flag is set;
                    move the READ BYTE to DATA BUFFER;
                    clear BIT/BYTE READY flag;
                    update WORD COUNTER ;
                    update DATA BUFFER POINTER
                until WORD COUNTER = 0;
        set BIT COUNTER to 16;/*To read the two CRC bytes*/
        repeat
             loop until BIT/BYTE READY flag is set;
             clear BIT/BYTE READY flag;
             update BIT COUNTER;
        until  BIT COUNTER = 0;
        mask OVERFLOW INTERRUPT REQUEST;
        check TAPE READ STATUS;
        if TAPE READ STATUS is good then
           set MOVING IN STOP GAP OP flag
        else set READ ERROR INCOMPLETE OP flag;
        output MOVE TAPE IN ERASE MODE control word;
        set PROGRAMMABLE INTERVAL TIMER to STOP GAP TIME
    end;/*TIMERPROCESS1*/
procedure TIMERPROCESS3;
/*This routine is executed when the MPU is interrupted*/
/*by the INTERVAL TIMER when the tape is moving in */
/*STOP GAP */
    begin
        check TAPE ENDING STATUS;
        if TAPE ENDING STATUS is good then
            begin
                set READ RECORD DONE OP flag;
                update RECORD COUNT FROM BOT
            end
        else set ENDING READ ERROR OP flag;
        output STOP TAPE control word
    end;
```

contd. Table 3.2.3

```
    begin/*READ PROCESS*/
        initialize DATA BUFFER POINTER;
        check TAPE READ STATUS;
        if TAPE READ STATUS is good then
           begin
               output MOVE FORWARD AT RD-WRT SPEED IN ERASE
                      MODE control word;
               set MOVING IN START GAP OF flag;
               set PROGRAMMABLE INTERVAL TIMER to START GAP TIME;
               save STACK POINTER
    end;/*READPROCESS*/
```

Table 3.2.3 READ Routine of TAPE DEVICE.

```
procedure WRITERECORD;
    function TAPEWRTSTATUS;
       begin
           input TAPE STATUS from PORT A of PPI#2;
           mask out unwanted bits;
           test AVAILABLE,CASSETTE-IN-PLACE,READY,IN SYNC
                and CRC ERROR
       end;
    procedure TIMERPROCESS1;
    /*This procedure is executed when the MPU is interrupted*/
    /*by the INTERVAL TIMER when tape is moving in STARTGAP*/
        procedure TIMERPROCESS2;
        /*This routine is executed when the MPU is */
        /* interrupted by the INTERVAL TIMER when time*/
        /*required to write a record elapses*/
            begin
                output STOP TAPE control word;
                set WRITE ERROR INCOMPLETE OP flag;
                restore STACK POINTER
            end;
```

contd. Table 3.2.4

```
procedure BOTEOTPROCESS;
/*This routine is executed when the MPU is */
/*interrupted by the BOT-EOT TIMEOUT signal*/
    begin
        set EOT SEEN OP flag;
        output STOP TAPE control word
    end;
procedure OVUNDFLOWPROCESS;
/*This routine is executed when the MPU is */
/*interrupted by the UNDERFLOW SIGNAL*/
     begin
         output STOP TAPE control word;
         set UNDERFLOW OP flag
     end;
  procedure WRITEWORD;
  /*This routine writes a byte word,stored in */
  /*memory location CURRENT BUFFER,in bit serial*/
  /*form*/
      begin
          set BITCOUNTER to 8;
          repeat
              loop until BIT REQUEST flag is set;
              move next DATA BIT to PORT A of PPI#1;
              reset BIT REQUEST flag;
              update BIT COUNTER
          until BIT COUNTER = 0
      end;
  begin/*TIMERPROCESS1*/
      set WRITE IN PROGRESS OP flag;
      set INTERVAL TIMER to RECORD LENGTH TIME;
      /*to abort operation if write operation is */
      /*not completed soon enough*/
      loop until BIT REQUEST flag is set;
      move PREAMBLE BYTE to CURRENT BUFFER;
      enable UNDERFLOW INTERRUPT REQUEST;
      enable READ MODE;
      if BIT SERIAL DATA TRANSFER then
         call WRITEWORD
       else
          begin
              loop until BYTE REQUEST flag is set;
              move PREAMBLE BYTE to PORT A of PPI#1;
              reset BYTE REQUEST flag
          end;
```

contd. Table3.2.4

```
move DATA BYTE from DATA BUFFER to CURRENT
     BUFFER;
increment DATA BUFFER POINTER;
enable CRC GENERATOR;
if BIT SERIAL DATA TRANSFER then
   repeat
         call WRITEWORD;
         update WORDCOUNTER;
         if WORDCOUNTER<>0 then
            begin
                move DATA BYTE from DATA BUFFER
                    to CURRENT BUFFER;
                increment DATA BUFFER POINTER;
            end
   until WORDCOUNTER=0
else
    repeat
         loop until BYTE REQUEST flag is set;
         move contents of CURRENT BUFFER to
              PORT A of PPI#1;
          reset BYTE REQUEST flag;
          update WORDCOUNTER;
          if WORDCOUNTER <> 0 then
             begin
                 move next DATA BYTE from DATA
                   BUFFER to the CURRENT BUFFER;
                 increment DATA BUFFER POINTER
             end
     until WORDCOUNTER = 0;
     set BIT COUNTER to 16;/*To read two CRC */
                           /*BYTES*/
     repeat
          loop until BIT REQUEST flag is set;
          enable SHIFT CRC;
          reset BIT REQUEST flag;
         update BITCOUNTER
     until BIT COUNTER=0;
     move POSTAMBLE BYTE to CURRENT BUFFER;
     disable SHIFT CRC;
     call WRITEWORD;/*WRITE POSTAMBLE*/
      loop until BIT/BYTE REQUEST flag is set;
      decrement RECORD COUNTER;
```

contd. Table 3.2.4

```
                    if RECORD COUNTER = 0 then
                       begin/*No more records to write*/
                           output STOP TAPE control word;
                           set ERASE-STOP IN PROGRESS OF flag;
                           mask UNDERFLOW INTERRUPT REQUEST;
                           set PROGRAMMABLE INTERVAL TIMER to
                               STOP GAP TIME;
                       end
                    else
                        begin
                            input the TAPE STATUS from PORT A
                                  of PPI#2;
                            check for EOT SEEN;
                            if EOT SEEN then
                               begin
                                   output MOVE TAPE IN
                                     ERASE MODE control
                                       word;
                                   set WRITE ERASE IN PROGRESS
                                        OF flag;
                                   set the PROGRAMMABLE
                                    INTERVAL TIMER to STOPGAP
                                     TIME;
                               end
                            else
                                 begin
                                     output STOP TAPE control
                                      word;
                                      set ERASE STOP IN
                                       PROGRESS OF flag;
                                      mask UNDERFLOW INTERRUPT;
                                      set PROGRAMMABLE INTERVAL
                                         TIMER to STOPGAP TIME;
                                 end
                        end
        end;/*TIMERPROCESS1*/
procedure TIMERPROCESS3;
/*This routine is executed when the MPU is interrupted*/
/*by the INTERVAL TIMER when tape is moving in STOPGAP*/
    begin
        increment RECORD COUNT from BOT;
        test OP flag;
        if ERASE STOP IN PROGRESS then
           begin
               set RECORD COMPLETE-STOP OP flag;
               output STOP TAPE control word;
               mask all INTERRUPT REQUESTS
           end
        else set RECORD COMPLETE-ERASE GO OP flag
    end;
```

contd. Table 3.2.4

```
begin/*WRITERECORD*/
     initialize DATA BUFFER POINTER;
     check TAPE STATUS;
     if TAPE STATUS is good then
        begin
             set MOVING IN STARTGAP OF flag;
             output MOVE TAPE IN ERASE MODE control word;
             set PROGRAMMABLE INTERVAL TIMER to STARTGAP TIME;
             save STACK POINTER
        end
     else set WRITE ERROR IN ENDING STATUS OF flag
end;/*WRITERECORD*/
```

Table 3.2.4 WRITE Routine of TAPE DEVICE.

upto 24 K bits/sec in Bit Serial Form and upto 5K bytes/sec. (40K bits/sec) in Byte Parallel Form.(assuming one microsecond CPU clock period). For lower speeds of the device, the Bit Serial Form is chosen and for higher speeds the Byte Parallel form is chosen. The type of data transfer is conveyed to the Read/Write Routines by the parameter DATARATERANGE .

3.3. FLOPPY DISK CONTROLLER :

The floppy disk is a removable magnetic storage media which is permanently contained in a paper envelope. The diskette drive is a low cost peripheral which performs the electro-mechanical and Read/Write functions necessary to record and recover data on the diskette. The floppy disk is divided into several concentric tracks and the data is recorded serially on each track. Normally, single R/W Head is used for transferring the data and each track is made available to the R/W Head by accessing the Head with a stepper motor and carriage assembly.

3.3.1 RECORDING FORMAT : Each data bit recorded on the diskette has an associated clock bit recorded with it. Data written on and read back from the diskette takes the form as shown in Figure 3.3.1. The binary data pattern shown represents a 101.

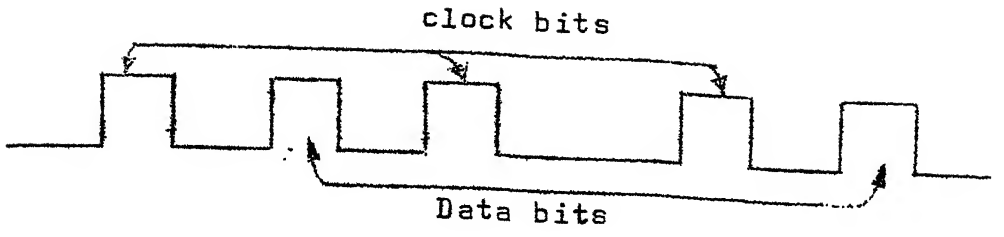


Figure 3.3.1 Data pattern.

The clock bits and data bits (if present) are interleaved as shown in the above figure.

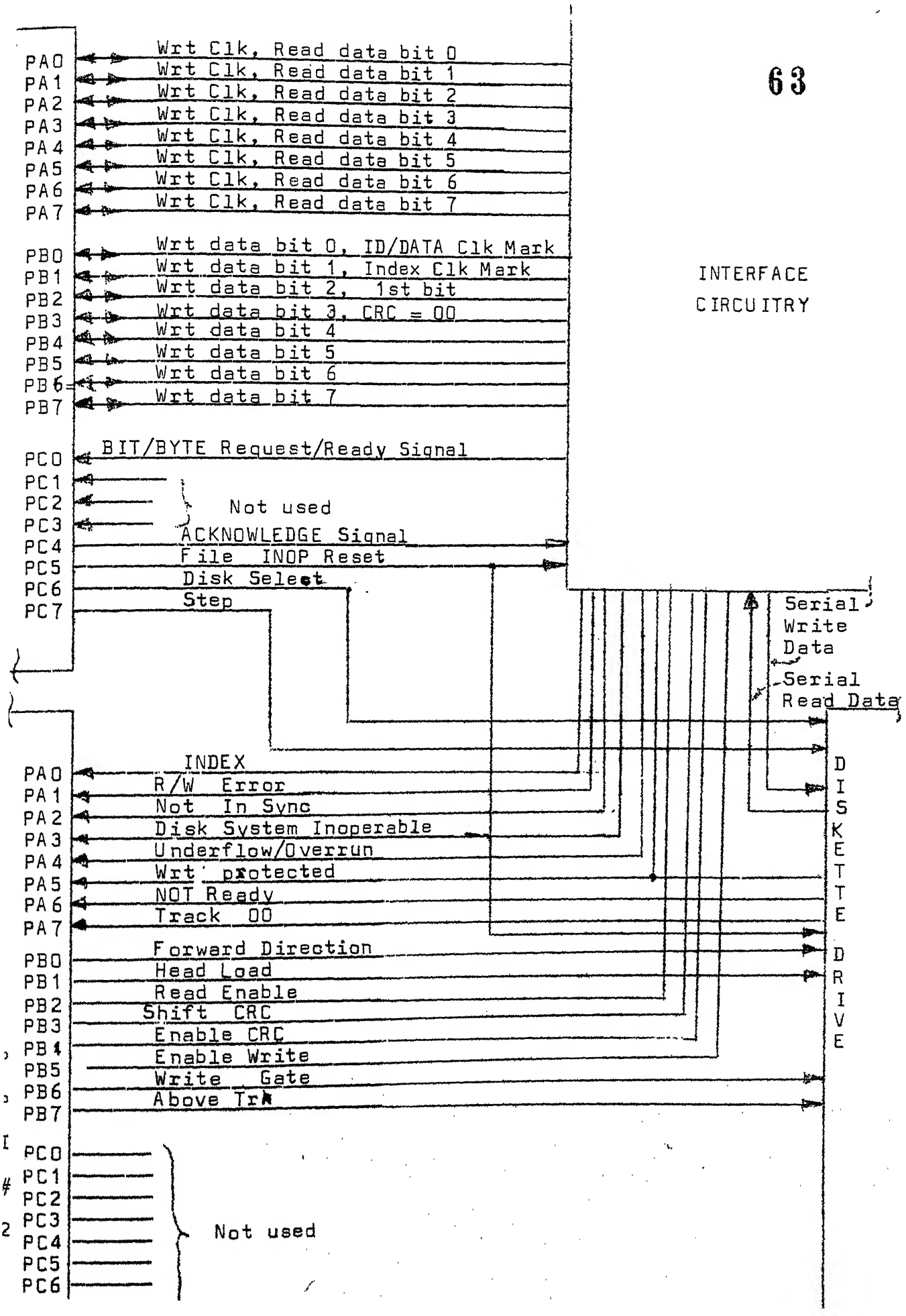
3.3.2 TRACK FORMAT : Tracks may be formatted in numerous ways and are dependent on the using system. Index Recording Format is used here. In this format, the system may record one long record or several smaller records. Each track is started by a physical index pulse and then each record is preceded by a unique recorded identifier.

3.3.3 Functions of Floppy Disk Controller: The following basic routines are described in this section.

1. SEEK ROUTINE
2. READ A RECORD and
3. WRITE DATA RECORD.

3.3.4 INTERFACE SIGNALS: The functional block diagram which shows all the signal lines between the floppy disk system and the PPIs is shown in Figure 3.3.2.

PPI # 1 is used primarily as a data buffer. PORT A is the WRITE CLOCK buffer register


PA0
PA1
PA2
PA3
PA4
PA5
PA6
PA7
Wrt Clk, Read data bit 0
Wrt Clk, Read data bit 1
Wrt Clk, Read data bit 2
Wrt Clk, Read data bit 3
Wrt Clk, Read data bit 4
Wrt Clk, Read data bit 5
Wrt Clk, Read data bit 6
Wrt Clk, Read data bit 7
PB0
PB1
PB2
PB3
PB4
PB5
PB6
PB7
Wrt data bit 0, ID/DATA Clk Mark
Wrt data bit 1, Index Clk Mark
Wrt data bit 2, 1st bit
Wrt data bit 3, CRC = 00
Wrt data bit 4
Wrt data bit 5
Wrt data bit 6
Wrt data bit 7
INTERFACE
CIRCUITRY
PC0
PC1
PC2
PC3
PC4
PC5
PC6
PC7
BIT/BYTE Request/Ready Signal
Not used
ACKNOWLEDGE Signal
File INOP Reset
Disk Select
Step
Serial
Write
Data
Serial
Read Data
PA0
PA1
PA2
PA3
PA4
PA5
PA6
PA7
INDEX
R/W Error
Not In Sync
Disk System Inoperable
Underflow/Overrun
Wrt protected
NOT Ready
Track 00
PB0
PB1
PB2
PB3
PB4
PB5
PB6
PB7
Forward Direction
Head Load
Read Enable
Shift CRC
Enable CRC
Enable Write
Write Gate
Above Trk
DISKETTE DRIVE
PC0
PC1
PC2
PC3
PC4
PC5
PC6
Not used

for write operations and the READ DATA gate for read operations. PORT B is the WRITE DATA buffer for write operations and is used as status signal gate for read operations. The four read status signals which are routed to PPI # 1, PORT B are :

	PB 0	ID/DATA CLOCK MARK
	PB1	INDEX CLOCK MARK
	PB2	1st Bit
	PB3	CRC = 00

PORT A and PORT B of PPI # 1 are programmed to be INPUT during Read operation and to be OUTPUT during Write operation. The control and status signals are interfaced to the MPU system bus through PPI # 2. The control signals are :

PPI # 2	PB0	FORWARD DIRECTION
	PB1	HEAD LOAD
	PB2	READ ENABLE
	PB3	SHIFT CRC
	PB4	ENABLE CRC
	PB5	ENABLE WRITE
	PB6	WRITE GATE
	PB7	ABOVE TRK
PPI # 1	PC5	FILE INOP RESET
	PC6	DISK SELECT
	PC7	STEP

The Status signals are :

PPI # 2	PA0	INDEX
	PA1	R/W ERROR
	PA2	NOT INSYNC
	PA3	DISK SYSTEM INOPERABLE
	PA4	UNDERFLOW/OVERRUN
	PA5	WRITE PROTECTED
	PA6	NOT READY
	PA7	TRACK 00

TRACK 00 status signal is generated, when the R/W head is positioned on outermost track of the diskette. The R/W Error is the logical OR of NOT INSYNC, DISK SYSTEM INOPERABLE, and UNDERFLOW/OVERRUN.

The interrupt signals are :

IR0	INTERVAL TIMER INTERRUPT
IR1	R/W ERROR
IR2	READY
IR3	INDEX.

3.3.5 EXTERNAL HARDWARE DESCRIPTION:-

Read Operation Interface: The SERIAL READ DATA signal contains both clock and data information. The hardware contains a Data and clock recovery block whose function is to produce a separated NRZ data, a clock that is synchronized to the data and serial NRZ clock information and another clock that is synchronized to the

recovered clock pattern from the SERIAL READ DATA. The interface contains a serial to parallel shift register, a read data buffer register, a bit counter (used to determine the byte boundaries) and a CRC polynomial generator (used for detection of read errors). The interface also contains a clock shift register and a decode of the clock portion of the data, ID and index address marks.

Write Operation Interface :- This interface converts the parallel write clock and write data into interleaved serial data.

Error Detect Logic block : This block contains error latches. When an error condition is trapped, the appropriate error latch will be set.

3.3.7 SOFTWARE DESCRIPTION :-

The basic control functions are :

1. INITIALIZATION. 2. SEEK operation . 3. READ a record and 4. WRITE a given number of records.

1. INITIALIZATION :- This routine is used to initialize the peripheral interface devices during system power-up. A detailed explanation of this routine is given in Table 3.3.1.

2. SEEK Operation :- The seek function is divided into two parts, the SEEK INITIALIZATION routine and the INTERRUPT DRIVEN SEEK. The SEEK INITIALIZATION

routine calculates the number of tracks and direction the R/W head must move, sets up the disk control signals, and generates the first interval timer interrupt to begin head movement.

Head movement from track to track is controlled by the INTERRUPT DRIVEN SEEK program routine. This routine is interrupt driven by the interval timer. The two control signals DIRECTION And STEP govern head movement from track to track whenever a STEP pulse is generated, the R/W head will be moved to the next track. When the seek is completed, a seek complete indicator flag is set.

The RESTORE operation is similar to the seek operation. The main difference between seek and restore is that a restore operation always moves the R/W head to track 00 (outermost track).

A detailed explanation of the SEEK routine is given in Table 3.3.2.

3. READ routine :- This routine is used to read both ID and DATA records. The differences between an ID and data field operation:

1. The number of bytes in the field.
    a. ID field = 7 bytes
    b. Data field (fixed format) = 131 bytes.

```
ocedure INITIALIZE;
  besin
       define PORT A,PORT B, and PORT C(LOWER) of PPI#1
       as inputs and PORT C(UPPER) of PPI#1 as output;
       define PORT A,PORT C(LOWER) of PPI#2 as inputs
       and PORT B, PORT C(UPPER) of PPI#2 as outputs;
       reset ACNOWLEDGE,FILE INOP,DISK SELECT and STEP flass;
       initialize PROGRAMMABLE INTERVAL TIMER;
       initialize PROGRAMMABLE INTERRUPT CONTROLLER;
       mask all INTERRUPT REQUESTS
  end;
```

able 3.3.1 INITIALIZATION Routine of FLOPPY DISK.

```
ocedure SEEKINIT;
  function DISKSEEKSTATUS;
       besin
            input DISKSTATUS;
            check DISKINOPERABLE,READY, and HEAD LOAD
       end;
  rrocedure TIMERPROCESS1;/*This routine is executed when*/
    /*the MPU is interrupted by the INTERVAL TIMER when the */
   /*time required to move R/W HEAD from one track to next*/
   /*track elapses*/
       besin
            check DISK SEEK STATUS;
            if DISK SEEK STATUS is sood then
               besin
                    decrement TRACK DELTA;/*TRACK DELTA is the*/
                     /*difference between the TARGET TRACK and*/
                     /*CURRENT TRACK*/
                    if TRACK DELTA = - 1 then
                        besin/*SEEK operation is over*/
                             mask all INTERRUPT REQUESTS;
                             if RESTORE OPERATION then
                                  set RESTORE ABORTED OP flas
                             else set SEEK COMPLETED OP flas
                        end
                    else
```

contd. Table 3.3.2

```
                begin
                    update CURRENT TRACK buffer;
                    if RESTORE OPERATION then
                       begin
                            input DISK STATUS;
                            test for TRACK00;
                             if TRACK00 then
                                 begin
                                     move zero to
                                     CURRENT TRACK buffer;
                                     move OFFH to TRACK
                                                  DELTA;
                                     mask all INTERRUPT
                                              REQUESTS;
                                      set RESTORE
                                       COMPLETED OP flag;
                                 end
                            else
                               begin
                                   generate STEP pulse;
                                   set PROGRAMMABLE
                                     INTERVAL TIMER to
                                     TRACK INTERVAL
                                      TIME
                               end
                          end/*RESTORE OPERATION*/
                       else
                           begin/*SEEK OPERATION*/
                               generate STEP pulse;
                               set INTERVAL TIMER to
                                TRACK INTERVAL TIME
                           end
                  end/*TRACK DELTA # -1 */
          end/*DISK STATUS is good */
     else
          begin
               move FFH to TRACK DELTA;

               if RESTORE OPERATION then
                  set RESTORE ABORTED OP flag
               else set SEEK ABORTED OP flag;
               mask all INTERRUPT REQUESTS
          end
end;/*TIMERPROCESS1*/
```

contd. Table 3.3.2

```
begin/*SEEKINIT*/
    if RESTORE OPERATION then
       begin
            initialize TARGET TRACK to zero;
            initialize CURRENT TRACK to NUMBER OF TRACKS
                       DISK + 10;
            set RESTORE OPERATION IN PROGRESS OP flag
       end
    else set SEEK IN PROGRESS OP flag;
    move difference of TARGET TRACK and CURRENT TRACK to
      TRACK DELTA;
    if TRACK DELTA < 0 then
       set FORWARD SEEK control flag
    else
       begin
            negate TRACK DELTA and CURRENT TRACK;
            set REVERSE SEEK control flag
       end;
    set DISK SELECT flag;
    generate FILE INOP RESET pulse;
    set PROGRAMMABLE INTERVAL TIMER to 3 milliseconds;
    /*To generate dummy interrupt and start SEEK operation*/
    enable INTERVAL TIMER INTERRUPT and READY INTERRUPT
end;/*SEEKINIT*/
```

Table 3.3.2 SEEK Routine of FLOPPY DISK

```
procedure READRECORD;
    function DSKRDSTATUS;
        begin
            input DISK STATUS;
            test NOT INSYNC,DISK SYSTEM INOPERABLE,OVERRUN,
                 NOT READY and NOT HEAD LOAD;
        end;
    procedure TIMERPROCESS;
    /*This procedure is executed when the MPU is interrupted*/
    /*by the INTERVAL TIMER when the time required to read a */
    /*record elapses */
        begin
            set READ ABORT BY INTERRUPT OP flag;
            output STOP READ control word;
            mask all INTERRUPT REQUESTS
        end;
```

contd. Table 3.3.3

```
begin/*READRECORD*/
     set READ IN PROGRESS OF flag;
     reset all control flags;
     set DISK SELECT flag;
     generate FILE INOP RESET pulse;
     fetch DISK READ STATUS and store in ERRORSTATUS;
     if ERRORSTATUS=0 then
        begin
             save STACK POINTER;
             initialize DATA BUFFER POINTER;
             enable all INTERRUPT REQUESTS;
             set PROGRAMMABLE INTERVAL TIMER to READ TIME;
             /*Time required to read a RECORD*/
             reset BYTE READY flag;
             enable CRC GENERATOR;
             repeat .
                  loop until BYTE READY flag is set;
                  increment DATA BUFFER POINTER;
                  move DATA BYTE from PORT A of PPI#1
                  to DATA BUFFER;
                  reset BYTE READY flag;
                  update BYTE COUNTER;
             until BYTE COUNTER=0;
             loop until BYTE READY flag is set;
             /*Wait for first CRC BYTE*/
             reset BYTE READY flag;
             loop until BYTE READY flag is set;
             /*wait for second CRC BYTE */
             reset BYTE READY flag;
             output STOP READING control word;
             fetch CRC STATUS and store in ERRORSTATUS;
        end;/*ERROR STATUS =0*/
     if ERRORSTATUS=0 then set READ COMPLETE OF flag
     else set READ ABORT BY PROGRAM OF flag;
end;/*READRECORD*/
```

Table 3.3.3 READ Routine of FLOPPY DISK.

```
procedure WRTDATAREC;
     function DSKWRTSTATUS;
          begin
               fetch DISK STATUS;
               test INDEX,NOT INSYNC,DISK SYSTEM INOPERABLE,
                    UNDERFLOW,WRITE PROTECTED,READY and
                    HEAD LOAD;
          end;
     procedure TIMERPROCESS;
     /*This procedure is executed when the MPU is interrupted*/
     /*by the INTERVAL TIMER when the time required to write */
     /*a record elapses */
        begin
            set WRITE ABORT BY INTERRUPT OP flag;
            output STOP WRITE control word;
            mask all INTERRUPT REQUESTS
        end;
   begin/*WRTDATAREC*/
        set DISK SELECT flag;
        generate FILE INOP RESET pulse;
        check DISK WRITE STATUS;
        if DISK WRITE STATUS is good then
           begin
               save STACK POINTER;
               set WRITE IN PROGRESS OP flag;
               select PORT A and PORT B of PPI#1 as outputs;
               /*For write operation */
               move GAP CLOCK PATTERN to PORT A of PPI#1;
               move GAP DATA PATTERN to PORT B of PPI#1;
               set ENABLE WRITE flag;
               if CURRENT TRACK > ABOVE TRACK then
               /*ABOVE TRACK is that track beyond which the */
                /*WRITE CURRENT amplitude should be decreased*/
                  set ABOVE TRACK flag;
               set BYTE COUNTER to GAP BYTE COUNT;
               set INTERVAL TIMER to WRITE TIME;/*Time required*/
                 /*to write a completed record */
               set WRITE GATE OF flag;
               clear BYTE REQUEST OP flag;
               repeat /*writes GAP BLOCK */
                   loop until BYTE REQUEST flag is set;
                   reset BYTE REQUEST flag;
                   update BYTE COUNTER;
               until BYTE COUNTER = 0;
               loop until BYTE REQUEST flag is set;
```

contd. Table 3.3.4

```
            move ADDRESS CLOCK PATTERN to PORT A of PPI#1;
            move ADDRESS MARK BYTE to PORT B of PPI#1;
            enable CRC GENERATOR;
            reset BYTE REQUEST OP flag;
            set BYTE COUNTER to 128;/*Fixed format:Each */
            /*record contains 128 bytes excluding CRC */
            /*bytes */
            initialize DATA BUFFER POINTER;
            repeat  /*writes data portion */
                loop until BYTE REQUEST flag is set;
                move DATA CLOCK PATTERN to PORT A of PPI#1;
                increment DATA BUFFER POINTER;
                move the next DATA BYTE from DATA BUFFER to
                 PORT B of PPI#1;
                reset BYTE REQUEST OP flag;
                update BYTE COUNTER;
            until BYTE COUNTER = 0;
            set SHIFT CRC control flag;
            loop until BYTE REQUEST flag is set;
             /*wait for first CRC BYTE */
            reset BYTE REQUEST flag;
            move "00" DATA PATTERN to PORT B of PPI#1;
             /*Preload GAP DATA PATTERN */
            loop until BYTE REQUEST flag is set;
            /*wait for second CRC BYTE */
            reset BYTE REQUEST flag;
            reset SHIFT CRC control flag;
            loop until BYTE REQUEST flag is set;
            /*wait for GAP BYTE */
            reset BYTE REQUEST flag;
            mask all INTERRUPT REQUESTS;
            reset WRITE GATE control flag;
            select PORT A and PORT B of PPI#1 as inputs;
              /*Prepare for READ operation */
            reset ENABLE WRITE control flag;
            set WRITE COMPLETE OP flag;
        end /*DISK WRITE STATUS is good*/
    else set WRITE ABORT BY PROGRAM OP flag;
    output STOP WRITING control word;
    mask all INTERRUPT REQUESTS;
end; /* WRTDATAREC */
```

Table 3.3.4 WRITE Routine of FLOPPY DISK.

compiler is supplied by the INTEL Corporation. The object code of the compiler is the INTEL 8080's assembly language. The hardware modules are not developed and the software is tested on the INTEL 8080 SIMULATOR. The software control modules are evaluated in the next chapter.

CHAPTER 4

EVALUATION OF CONTROL ROUTINES

For successful data transfer between MPU and the peripheral device, it is necessary to synchronize the data transfer program to the peripheral because the peripheral data clock is asynchronous with respect to the program clock. The I/O controller which handles the data transfer consists of both hardware and software. An implicit assumption is that the best trade-off occurs by minimizing the hardware in the controller.

In a microprocessor based I/O controller, it is necessary to determine :

1. How fast can the microcomputer transfer program move data (as contrasted with a direct memory access scheme) ?
2. Will a given data transfer program work successfully in the system ?
3. Is there any processing time remaining after handling the data movement ?
4. Can any additional time dependent functions be performed ?
5. What is the maximum length routine that can be performed in addition to the data transfer ?

An analysis is required that will provide a technique for testing the operation of a proposed program. In addition,

if there is unused processing time in the system, it may be possible to eliminate additional hardware (e.g. buffer registers). If a given program does not work in the system, the analysis should enable the user to modify the program or add additional hardware to allow the system to work .

When a peripheral signals the MPU requesting processing time, it will be referred to as a Service Request (SR). Read or write Data Transfers are both examples of such service requests and wherever the term Read Data Transfer is referred, it is meant to both Read and write Data transfers.

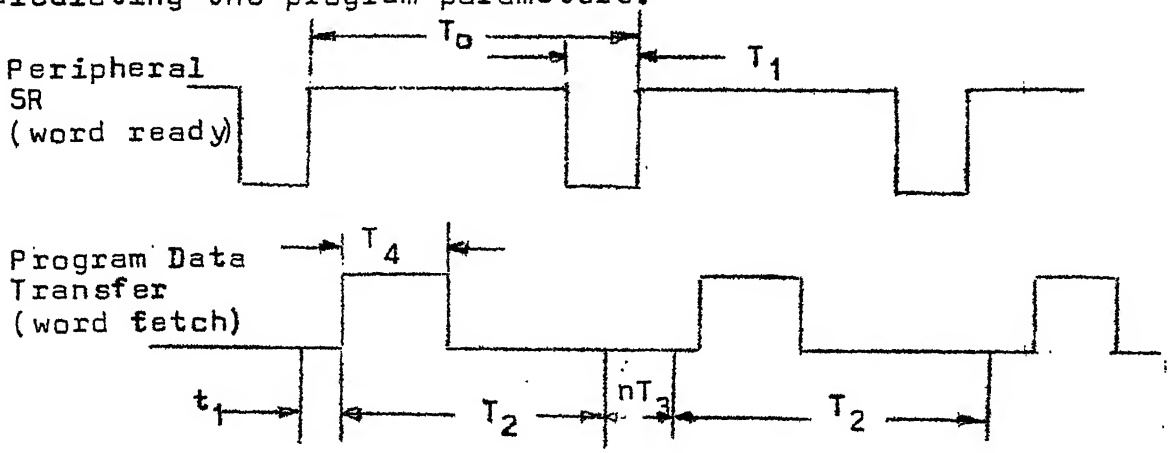
4.1 SERVICE REQUESTS AND PROGRAMS AS WAVEFORMS ON A TIMING DIAGRAM :

The process of synchronizing a data transfer program to a peripheral can be visualized more easily when the SR's and the program are both represented as waveforms on a timing diagram. The peripheral SR waveforms is developed from the specifications of the peripheral which identify the maximum time, T_1, it takes to load the data buffer (the period during which data is invalid), and the minimum period, T_o, between service requests. The period T_o is the worst case (fastest) peripheral word data rate.

The data transfer waveform is developed by writing the actual data transfer program and then calculating the time it takes to :

1. Capture the data (T_4)
2. Process the data (T_2- includes period T_4)
3. Loop in a synchronization delay loop until a SR is active (n T_3- where T_3 is the single loop time and n is the number of times the program loops).

These values are calculated by counting the number of processor clock cycles required to excute each function, and multiplying the numbers by the MPU clock rate. The waveforms and notation for a typical situation are illustrated in Figure 4.1.1 . Figure 4.1.2 shows a flow chart for a data transfer program. Figure 4.1.3 details the technique for calculating the program parameters.



T_o = Period of service request (word ready period)

Contd... Fig. 4.1.1

T_1 = Service request update time (Data Invalid)

T_2 = Program processing time (Includes time to capture the data)

T_3 = Synchronization loop time when the program has checked and found no active service requests.

T_4 = Data capture time

t_1 = Initial offset between the SR and program Data Transfer waveforms.

n = number of times the program goes through the synchronization delay loop.

Fig. 4.1.1 Peripheral Service Request (SR) and Data Transfer Program Waveforms and Notation.

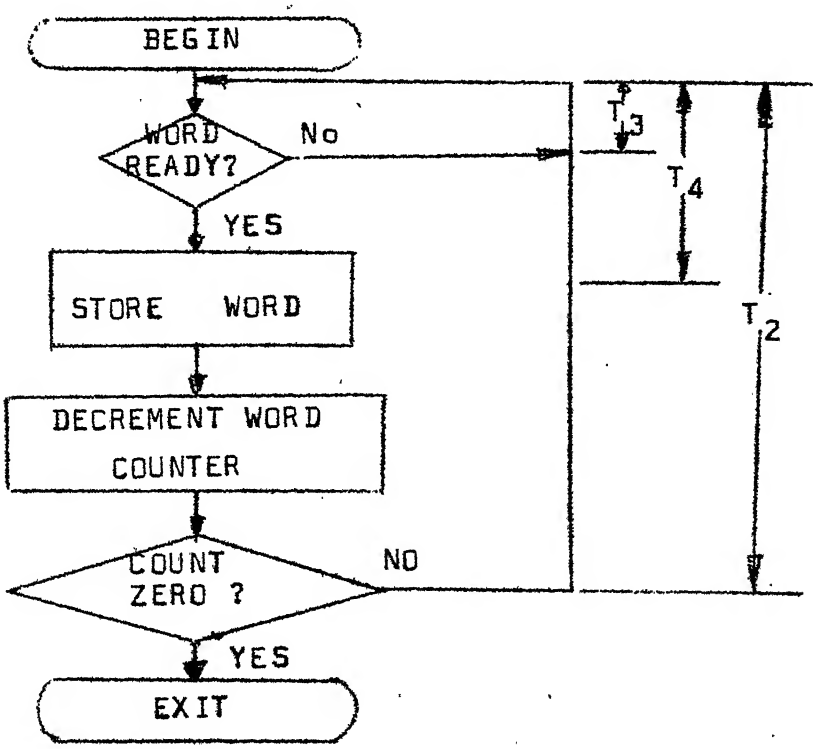


Figure 4.1.2 : Flow Chart for a Typical Data Transfer Program

LABEL	MNEMONIC	OPERAND	COMMENT		PROCESSOR CYCLES
RDLOOP	IN	PORT C	INPUT CONTENTS OF PORT C IN TO ACCUMULATOR	SYNC LOOP (T_3)	10
	ANI 01H		MASK OUT ALL BITS EXCEPT THE BYTE READY BIT		7
	SUI 00H		TEST FOR ZERO		7
	JZ	RDLOOP	LOOP IF SR IS INACTIVE		10
	LXI H	PTRADDR	LOAD ADDRESS OF DATA BLOCK POINTER IN TO HL REGISTER PAIR		10
	MOV C	PTRVALUE	MOVE THE POINTER VALUE INTO REGISTER C		7
	MVI B	00H	CLEAR B		7
	DCR H INR L		LOAD STARTING ADDRESS OF THE DATA BLOCK IN TO REGISTER PAIR HL		5 5
	DAD B		ADD THE POINTER VALUE (OFFSET VALUE) TO THE CONTENTS OF REGISTER PAIR HL (STARTING ADDRESS OF DATA BLOCK)		10
	XCHG		EXCHANGE CONTENTS OF REGISTER PAIRS HL AND DE	(T_4)	4
	IN	PORT A	INPUT READ DATA IN TO ACCUMULATOR		10
	STAX D		STORE THE DATA WORD IN THE DATA BLOCK		7
	MVI A OUT MVI A OUT	09H CWR55 08H CWR55	GENERATE THE ACKNOWLEDGE SIGNAL		7 10 7 10
					(T_2)

DATA TRANSMISSION TIMING

Contd..... Figure 4.1.3

	LXI	**H**	**PTRADDR**	LOAD ADDRESS OF DATA BLOCK POINTER IN TO REGISTER PAIR HL	10
	INR	M		INCREMENT THE VALUE OF THE POINTER	10
	MVI	L	F4H	UPDATE THE CONTENTS OF REGISTER PAIR HL TO POINT TO THE MEMORY LOCATION OF WORD COUNTER	7
	DCR	M		DECREMENT WORD COUNT	10
	MOV	A,M		MOVE THE VALUE OF WORDCOUNT IN TO ACCUMULATOR	7
	SUI	00H		TEST FOR ZERO	7
	JNZ	RDLOOP		IF WORD COUNT IS NOTZERO RETURN FOR NEXT WORD	10
EXIT	...				
				Total :	194

IF THE MPU CLOCK PERIOD IS 1 MICRO SECOND THEN

SYNC LOOP TIME, T_3 = 17 MICROSECONDS

PROGRAM PROCESSING TIME, T_2 = 194 MICROSECONDS

DATA CAPTURE TIME, T_4 = 82 MICROSECONDS

Figure 4.1.3 Data Transfer Program Indicating Method Used to Calculate Program Parameters.

4.2 DEVELOPMENT OF EQUATIONS AND INEQUALITIES USED TO TEST SUCCESSFUL SYSTEM OPERATION :

A successful data transfer means that each time the peripheral indicates, via an SR, that a word is available the program is able to capture the data before it is

replaced by the next data word. It is implied that the program is able to process the data between data word transfers. (The data processing includes storing the data in RAM and checking whether it was the last word that needed to be transferred) . Similarly for data transfers to the peripheral, the program must make the data word requested available before the succeeding request arrives. In other words, a successful data transfer consists of avoiding an overrun (during READ) and underflow (during WRITE).

If the SR is not active at the time that the program checks for a SR, (i.e., the data word is not ready), then the program goes into a synchronization (sync) loop , which causes a delay (T_3) . At the end of a sync, the program again checks for an active SR .

In the following analysis, it is assumed that the values of the parameters detailed in Figure 4.1.1 are at their worst case limits and are constant for simplicity.

For the system to transfer data successfully the average word processing time T_{AVG} must be approximately equal to the peripheral data word SR period, T_o

$$T_{AVG} \approx T_o \qquad (1)$$

More precisely stated, in the limit as the number of words transferred, p , approaches infinity, the average word processing time, T_{AVG}, is exactly equal to the byte cell period, T_o .

$$\lim_{P \to \infty} T_{AVG} = T_o \quad \text{- - - - - - -} \quad (2)$$

The time T_{AVG} consists of the program word processing time, T_2, and a time nT_3 while the program loops until the next word is ready.

$$T_{AVG} = T_2 + \frac{1}{P} \left(\sum_{P=1}^{\infty} n_p T_3 \right) \text{- - -} \quad (3)$$

Where n_p is the number of sync loops taken while waiting for p^{th} SR, and whose value may vary from 0 to n ($n = n_p$ maximum).

The program byte processing time, T_2, must be equal to or less than the SR period T_o, or else the program could not keep up with the word rate of the peripheral. Therefore,

$$T_2 \leqslant T_o \text{ - - - - -} \quad (4)$$

If the program loops n times in the sync delay loop before the next data word is ready, then equation (4) can be modified to read :

$$T_2 + (n-1)\, T_3 \leqslant T_o \quad \text{- - - - - -} \quad (5)$$

Also, the time $T_2 + n\, T_3$ must be greater than T_o so that the program may begin the transfer of the next word even if the offset, T_1, is equal to zero. This is true simply because the program loops until the next word SR is active, Hence,

$$T_o < T_2 + n\, T_3 \quad \text{- - - -} \quad (6)$$

Therefore, the peripheral word ready period is bounded by $T_2 + (n-1)\ T_3$ and $T_2 + n\ T_3$ for successful operation :

$$T_2 + (n-1)\ T_3 \leqslant T_o < T_2 + n\ T_3 \quad - - - \quad (7)$$

If $T_o = T_2$ then the program and peripheral are said to be synchronous. If $T_o > T_2$ then the offset T_1 gets smaller and smaller until it is negative or zero, which means that after the program has processed one word, the next word will not be ready. At this time, the program goes into the synchronization loop, and samples the peripheral word Ready line until the SR is again active.

The maximum value of the synchronization loop for which the system will work maybe determined from the following argument. Since the peripheral SR and the program are independent, it is entirely possible that the SR occurs immediately after the program has initiated a sync loop. Since the data capture time is T_4 and the data is invalid for a period T_1 out of every T_o, it is necessary that :

$$T_3 \leqslant T_o - T_1 - T_4 \quad - - \quad - - \quad - - \quad (8)$$

This is the inequality used to calculate the maximum permissible value of T_3 .

The parameters T_2, T_3 and T_4 are calculated for the data transfer programs of the three I/O controllers developed in the last chapter. These parameters are calculated following the method shown in Figure 4.1.3 and with the aid of

INTEL 8080 SIMULATOR. The SIMULATOR has a provision to calculate the number of MPU clock cycles required to execute a given segment of assembly program. The maximum data transfer rate which can be handled by the data transfer program corresponds to when $T_o = T_2$. If $T_o > T_2$ then the program goes into the synchronization loop and samples the Word Ready line until the SR is again active. The maximum number of sync loops, n, the program takes while waiting for the next SR is calculated by substituting the values of T_o, T_2 and T_3 in equation (7). The maximum value of sync delay, T_3, for which the system will work is determined by substituting the values of T_o, T_1 and T_4 in the inequality (8). The results, including storage requirements, are given in Table 4.1 .

Table 4.1

MPU CLOCK PERIOD= 1 MICRO SECOND

S.No.	Name of Device	Storage required (in bytes)	Type of data transfer	Timing Parameters (in Micro sec)					Successful data transfer?
				T_o	T_1	T_2	T_3	T_4	
1	Printer	0.9K	-	4000	1	3000	34	95	Yes
2	Tape Device	2.6K	BIT serial	83	1	337	34	220	No
			BYTE parallel	666	1	174	34	123	Yes
3	Floppy Disk system	1.1K	BYTE parallel	320	1	200	34	145	Yes

CHAPTER 5

CONCLUSIONS

An important part of the conclusions is a discussion on, to what extent the main objectives of the thesis have been fulfilled.

The first part of the work presents a method to develop microprocessor-based I/O controllers with minimum external hardware. The possible generalizations that can be made in the controller are explained in detail. It is observed that absolute generalization is not feasible due to the wide variations in the operating characteristics of the I/O devices. A generalized MPU system is proposed to handle different kinds of I/O devices and the possible generalizations in the external hardware and the software control modules are explained.

The second part of the thesis is the development of I/O controllers for the three devices : 1. Printer 2. Tape device and 3. Floppy disk system following the methodology developed in Chapter 2. All the software routines are developed in PL/M . The hardware modules are not developed and the software routines are tested on the INTEL 8080 SIMULATOR. The software control routines are generalized by passing the device speed parameters to the routines. A method to evaluate the software control routines is described.

All the developed software control routines are evaluated and checked their capability of handling high speed devices.

The generalization of an I/O controller has a drawback when the controller is dedicated to handle a particular I/O device. It is quite possible that that particular I/O device could be handled without the Hardware Timer (8253) and Programmable Interrupt Controller (8259). In that case, the cost of the system goes up. Usually, the MPU will be handling other tasks other than the I/O operation. The available processing time of the MPU and the unused data lines of the PPIs and Programmable Interrupt Controller can be utilized to handle the tasks other than the I/O operation. In that case, the effective cost of the system decreases.

The methodology that has been developed deals with handling one I/O device at a time. The methodology may be extended to handle several I/O devices simultaneously.

REFERENCES

1. Andrew S. Tanenbaum, "Structured computer organisation", Prentice-Hall of India, New Delhi, 1976 .

2. Blakeslee, T.R., "Digital Design with Standard MSI and LSI", Wiley, New York, 1975 .

3. Hilburn, J.L., and P.N. Julich., "Microcomputers/ Microprocessors; Hardware, Software, and Applications", Prentice-Hall, Inc., Englewood Cliffs, N.J., 1976.

4. "INTEL The 8080/8085 Microprocessor Book", Wiley-Interscience, John Wiley & Sons, New York, 1980 .

5. "INTEL 8008 and 8080 PL/M Programming Manual" ,INTEL Corprn., Santa Clara, California, October 1977 .

6. Ivan Flores, "COMPUTER LOGIC, The Functional Design of Digital Computers", Prentice-Hall, New Jersey, 1960.

7. "MICROPROCESSOR APPLICATIONS MANUAL", MOTOROLA Semiconductor Products Inc., McGraw-Hill Book Company,1975.

8. "MCS-80 USER'S MANUAL", Intel Corprn., Santa Clara, California, October 1977.

9. Peatman, John B., "Microcomputer Based Design", McGraw-Hill, Kogakusha, Tokyo, 1977 .

10. Pratt W.C., and F.H. Brown, "Automated Design of Microprocessor-Based Controllers", Industrial Electronics & Control Instrumentation, August 1975, Vol. IECI 22, No. 3, pp. 273-279

11. Rajaraman, V., and Radhakrishnan, T., "An Introduction to Digital Computer Design", Prentice-Hall of India, New Delhi, 1976.

12. "SOME REAL MICROPROCESSORS", AN INTRODUCTION TO MICROCOMPUTERS, Volume 2, OSBORNE/McGraw-Hill, Berkeley, California, August 1979 .

13. "SOME REAL SUPPORT DEVICES", AN INTRODUCTION TO MICROCOMPUTERS, Volume 3, Osborne & Associates, Inc., Berkeley, California, October 1978 .